Miskolci Egyetem
Gépészmérnöki és Informatikai Kar
Analízis Intézeti Tanszék

Szakdolgozat

Miskolci Egyetem

Alternatív grafikus megjelenítési módok

Készítette:

Vécsi Ádám

Gazdaságinformatikus BSc

Témavezető:

Piller Imre, egyetemi tanársegéd

Miskolc, 2018

Miskolci Egyetem
Gépészmérnöki és Informatikai Kar
Alkalmazott Matematikai Intézeti Tanszék **Szám:**

Szakdolgozat Feladat

Vécsi Ádám (IZBTF9) gazdaságinformatikus.

A szakdolgozat tárgyköre: képfeldolgozás, művészi szűrők

A szakdolgozat címe: Alternatív grafikus megjelenítési módok

A feladat részletezése:

Képfeldolgozási módszerek, azon belül a szűrők matematikai modelljének bemutatása.

Alternatív, absztrakt (például rajzfilm vagy festmény jellegű) megjelenítési módok elkészítése procedurális grafika és OpenCV segítségével. Alkalmazása képek és videók esetében. Tesztek készítése a szűrési eljárások futási idejének vizsgálatára.

Témavezetõ: Piller Imre (egyetemi tanársegéd)

A feladat kiadásának ideje: 2017. február 27.

. . . . . .
szakfelelõs

Eredetiségi Nyilatkozat

Alulírott **Vécsi Ádám**; Neptun-kód: `IZBTF9` a Miskolci Egyetem Gépészmérnöki és Informatikai Karának végzõs gazdasági informatikus szakos hallgatója ezennel büntetõjogi és fegyelmi felelõsségem tudatában nyilatkozom és aláírásommal igazolom, hogy *Alternatív grafikus megjelenítési módok* címû szakdolgozatom/diplomatervem saját, önálló munkám; az abban hivatkozott szakirodalom felhasználása a forráskezelés szabályai szerint történt.

Tudomásul veszem, hogy szakdolgozat esetén plágiumnak számít:

- szószerinti idézet közlése idézõjel és hivatkozás megjelölése nélkül;
- tartalmi idézet hivatkozás megjelölése nélkül;
- más publikált gondolatainak saját gondolatként való feltüntetése.

Alulírott kijelentem, hogy a plágium fogalmát megismertem, és tudomásul veszem, hogy plágium esetén szakdolgozatom visszautasításra kerül.

Miskolc, ... év ..... hó ..... nap

.....

Hallgató

1.

szükséges (módosítás külön lapon)

A szakdolgozat feladat módosítása

nem szükséges

....

dátum témavezetõ(k)

2. A feladat kidolgozását ellenõriztem:

témavezetõ (dátum, aláírás): konzulens (dátum, aláírás):

....

....

....

3. A szakdolgozat beadható:

....

dátum témavezetõ(k)

4. A szakdolgozat ..... szövegoldalt

..... program protokollt (listát, felhasználói leírást)

..... elektronikus adathordozót (részletezve)

.....

..... egyéb mellékletet (részletezve)

.....

tartalmaz.

....

dátum témavezetõ(k)

5.

bocsátható

A szakdolgozat bírálatra

nem bocsátható

A bíráló neve: .....

....

dátum szakfelelõs

6. A szakdolgozat osztályzata

a témavezetõ javaslata: .....

a bíráló javaslata: .....

a szakdolgozat végleges eredménye: .....

Miskolc, ...

a Záróvizsga Bizottság Elnöke

Tartalomjegyzék

1. Bevezetés **1**

2. Művészi jellegű szűrők **2**
2.1. Népszerű alkalmazási területek 2
2.1.1. Közösségi alkalmazások 2
2.1.2. Webkamera- és mobil alkalmazások 3
2.1.3. Filmek, játékok amelyekhez művészi szűrőket használtak 4
2.2. A megvalósítás nehézségei 5
2.3. Elvárások a szűrőkkel kapcsolatban 5

3. Matematikai eszközök **6**
3.1. Zaj szűréshez, elmosáshoz használatos szűrők 6
3.1.1. Átlagoló szűrő 6
3.1.2. Gauss szűrő 6
3.1.3. Medián szűrő 7
3.1.4. Kétoldalú szűrő 8
3.2. Éldetektálási módszerek 10
3.2.1. Sobel éldetektálás 11
3.2.2. Laplace éldetektálás 12
3.2.3. Canny éldetektálás 12
3.3. Szegmentálás 14
3.3.1. Mean shift algoritmus 14
3.4. Küszöbölés 14
3.4.1. Izodata algoritmus (Yanni) 15
3.4.2. Otsu algoritmus 16
3.4.3. Niblack algoritmus 17

4. Szűrő algoritmusok **18**
4.1. Cartoon-style filter 18
4.1.1. Gauss-piramis és kétoldalú szűrő 19
4.1.2. Színek konvertálása, medián szűrő 19
4.1.3. Adaptív küszöbölés az élek kiemelésére 20
4.1.4. A szűrőkkel és a küszöböléssel előállított képek egyesítése 21
4.2. Pencil sketch filter 21
4.2.1. Medián szűrő 22
4.2.2. Gauss szűrő 22
4.2.3. Az előző két lépés elosztása 23

4.2.4. Kontraszt széthúzása . . . 23
4.2.5. Vászon hozzáadása . . . 23
4.3. Cartoon filter . . . 25
4.3.1. Medián szűrő . . . 25
4.3.2. Laplace éldetektálás . . . 25
4.3.3. Küszöbölés . . . 26
4.4. Aquarelle-style filter . . . 27
4.4.1. Átlagoló szűrő . . . 28
4.4.2. Mean shift szegmentálás . . . 28

5. Implementáció C++ és OpenCV segítségével 30
5.1. OpenCV bemutatása . . . 30
5.2. Beépített művészi jellegű OpenCV-s szűrők . . . 30
5.2.1. Edge Preserving filter . . . 31
5.2.2. Detail Enhancing filter . . . 31
5.2.3. Pencil sketch filter . . . 32
5.2.4. Stylization filter . . . 32
5.3. Saját szűrők implementációja . . . 33
5.3.1. Cartoon-style filter . . . 34
5.3.2. Pencil sketch filter . . . 36
5.3.3. Cartoon filter . . . 38
5.3.4. Aquarelle-stlye filter . . . 39

6. Tesztek, eredmények bemutatása 40
6.1. Szűrők tesztelése képeken . . . 40
6.1.1. Cartoon-style filter . . . 41
6.1.2. Pencil sketch filter . . . 41
6.1.3. Cartoon filter . . . 42
6.1.4. Aquarelle-style filter . . . 42
6.2. Szűrők tesztelése videókon és valós időben . . . 43
6.2.1. Cartoon-style filter . . . 43
6.2.2. Pencil sketch filter . . . 43
6.2.3. Cartoon filter . . . 43
6.2.4. Aquarelle-style filter . . . 43

7. Összegzés 44

Irodalomjegyzék 46

1. fejezet

Bevezetés

A fényképek és videók készítése napjainkban már teljesen természetessé, megszokottá vált. Ennek köszönhetően népszerűvé váltak azok az eljárások és alkalmazások, melyek segítségével az eredeti felvételből valamilyen stilizált, művészi jellegű képet készíthetünk. Összefoglaló néven ezeket művészi szűrőknek nevezzük.

A dolgozat célja, hogy bemutassa ezen szűrők matematikai hátterét, megvalósítási módját, illetve saját készítésű szűrőket is felvonultasson.

Fiatalok körében eléggé elterjedtek a kép feldolgozással kapcsolatos alkalmazások, gondoljunk csak bele, hogy a napijainkban használatos közösségi alkalmazások mindegyikében megtalálható ez az opció. Akár arc felismeréssel képeken, valós időben és mozgóképeken, vagy akár előre kidolgozott szűrők segítségével, amik elmossák a színeket, kiemelik az éleket vagy valamilyen egyéb módon átalakítják a képeket. Ezekre is ki fogok térni a dolgozatomban.

Részletezem a különböző képfeldolgozási műveletek matematikai hátterét. Láthatók lesznek, a matematikai leírásoknál példákkal és ábrákkal illusztrálva.

Saját készítésű szűrőim rajzfilm, ceruza rajz szerű, valamint festmény jellegűek. A szűrőket képekkel lépésről-lépésre mutatom be, továbbá a C++ implementációját is részletezem.

Mint említettem szűrőimet C++ programozási nyelven szeretném megírni, OpenCV függvénykönyvtár segítségével. Az OpenCV az egy ingyenes, képfeldolgozási algoritmusokat tartalmazó könyvtár. A dolgozatom során ennek elemeire is kitérek.

A dolgozat végén tesztek segítségével megvizsgálom a saját készítésű szűrőket. A tesztek elsősorban a szűrők feldolgozási idejére, valamint az egyes lépések feldolgozási idejére vonatkoznak. A szűrők készítésénél szempont, hogy azokat ne csak statikus képekre, hanem videókra és kameraképre is lehessen alkalmazni. Ezek tesztelésére, egy képkocka per másodperc tesztet használok, amivel kiderül, hogy egy másodperc alatt hány képkockát tudunk előállítani az egyes szűrők esetén.

2. fejezet

Művészi jellegű szűrők

A mai világban elég népszerűek az olyan számítógépes vagy mobiltelefonos alkalmazások, amelyekkel fényképeket vagy videókat lehet átalakítani valós időben, képfeldolgozási módszerekkel. Dolgozatomban ezen alkalmazások működésének hátterével, jellemző algoritmusaival és megvalósítási módjukkal foglalkozom.

A művészi szűrők mögött komoly tudományos eredmények állnak. Többen foglalkoznak azzal, hogy egy-egy művészi jellegű hatás megvalósításának milyen lehetőségei vannak. Alapvetően nagyon irányzatként a témakörben az élek megtartásával történő szűrést [1], és a szegmentálásra épülő objektumkijelölést [2] különíthetjük el.

A következő szakaszok egyelőre azt vizsgálják, hogy milyen területeken találkozhatunk a művészi szűrők alkalmazásával a népszerű weboldalak és alkalmazások körében.

2.1. Népszerű alkalmazási területek

A fiatalok többsége ha képfeldolgozásról van szó egyből a közösségi média adta lehetőségekre gondol. Ilyen például az Instagram, Snapchat vagy a Messenger alkalmazások. Ezek az applikációk a telefonok, tabletek kameráját használja, mint a képek vagy videók forrását. Dolgozatomban ezek működésére is kitérek.

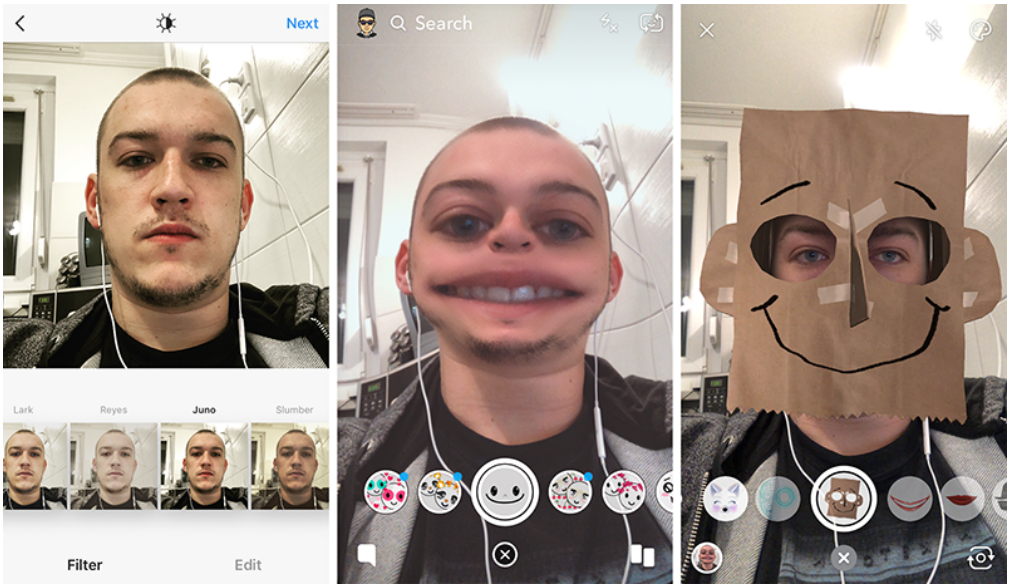
2.1.1. Közösségi alkalmazások

A közösségi alkalmazásokba előszeretettel építenek különféle képfeldolgozási módszereket. A 2.1. ábrán láthatunk egy-egy példát az Instagram, Snapchat és Messenger alkalmazások egy-egy szűrőjének alkalmazására. A következőkben ezen alkalmazások rövid áttekintésére kerül sor.

Instagram Valószínűleg az Instagram egyik legismertebb applikáció, amely képekkel és képfeldolgozással foglalkozik [3]. Ez az alkalmazás pontosan arra készült, hogy a felhasználók megoszthassák egymással ezeket, majd különféle művészi szűrőkkel láthassanak el. A szűrők köre folyamatosan bővül, valamit egyre testreszabhatók lesznek, például lehet állítani a képek dőlésszögén, fényességén, hogy mennyire legyen kontrasztos, a színek melegségén, mennyire legyen elmosva és hogy hol legyenek az elmosódások.

Snapchat A Snapchat is egy képfeldolgozással kapcsolatos alkalmazás [4], bár teljesen más a koncepció mint az Instagram esetében. Itt is a barátainkkal, imerőseinkkel tudjuk megosztani a grafikus tartalmainkat, de ezek a nyilvánosság számára nem láthatók, csak azok láthatják akiknek adunk jogosultságot, hogy láthassák és ők is csak egy bizonyos ideig, amit mi határozunk meg küldéskor. Ez az alkalmazás nem csak egyszerűen művészi szűrőkkel foglalkozik, hanem képes detektálni az arcokat. Mivel az alkamazás felismeri az arcunkat, így különböző tárgyakat/effekteket képes rárakni ennek megfelelően, például napszemüveget, vicces sapkákat, különböző fényeket, eltorzítja az arcunk vagy a szemünket.

Messenger Ez az alkalmazás nem kifejezetten képfeldolgozással foglalkozik, azonban már megjelentek benne ezek a funkciók is. Ebben az alkalmazásban barátainkkal, imerőseinkkel tudunk chatelni, tartalmakat megosztani, akár privát módon, akár csoportosan [5]. Képeinket hasonló módon tudjuk szerkeszteni, mint a Snapchat alkalmazással. Itt is arcfelismerés alapján tud rárakni az arcunkra a program mindenféle effektet. Ebben a programban elérhető a videóhívás is ahol szintén haszálhatók effektek.



2.1. ábra. Az Instagram, Snapchat és Messenger alkalmazások művészi szűrőiről készült képernyőfotók

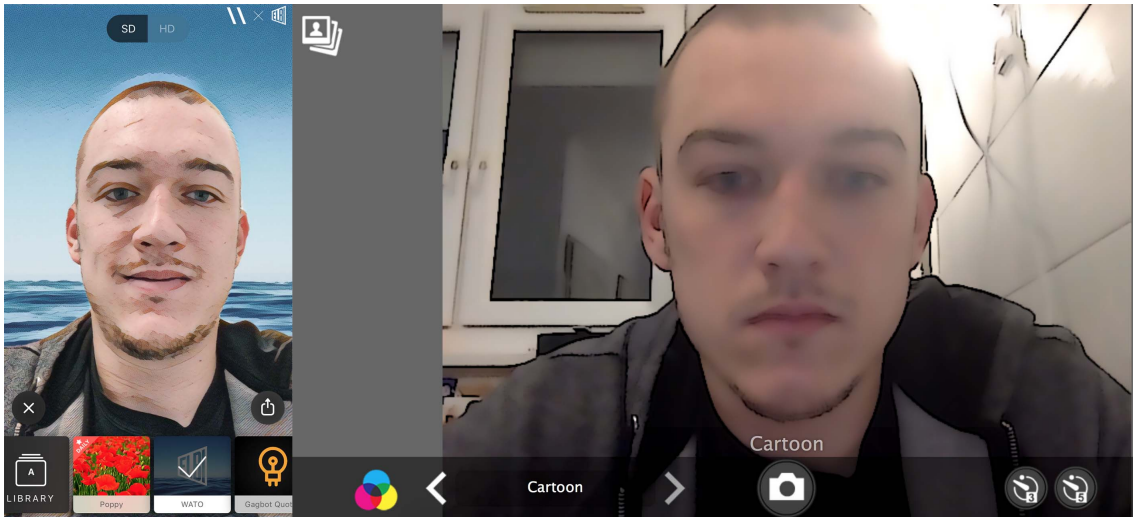
2.1.2. Webkamera- és mobil alkalmazások

A közösségi alkamazásokon kívül, van néhány olyan weboldal, mobil alkalmazás amely szintén komplikáltabb képfeldolgozási módszereket használ. A következőkben ezek közül a Prisma alkalmazást és a webkamerás alkalmazások néhány jellemzőjét mutatom be.

Prisma Ez az alkalmazás a képfeldolgozáshoz mesterséges intelligenciát, jellemzően neurális hálókat haszánál, hogy művészi szűrőkkel ellátott képeket hozzon létre [6]. A felhasználó az alkalmazásba betölti az átalakítandó fényképet, kiválasztja a számára

megfelelő szűrőt, pár másodperc múlva a szűrővel elláttott kép megjelenik az alkalmazásban. A képfeldolgozás a Prisma esetében távoli szerveren történik.

Webkamera alkalmazások Számtalan olyan program, webalkalmazás van amely tartalmaz képfeldolgozási funkciókat. Egy ilyen például a *Pixect* [7]. Léteznek továbbá gyári beépített programok, amelyek a webkamera saját illesztőprogramjában találhatók meg, és különböző szűrőket tudnak rárakni a képekre. Művészi szűrőkkel ellátott képekre láthatunk néhány példát a 2.2. ábrán.



2.2. ábra. Példa a Prisma és a `https://www.pixect.com` alkalmazások által készített művészi szűrőkkel ellátott képekre.

2.1.3. Filmek, játékok amelyekhez művészi szűrőket használtak

Művészi jellegű szűrőket nem csak magánfelhasználók használnak arra, hogy érdekesebbé tegyék az elkészült fényképeiket, videóikat, hanem ilyen filtereket használnak videójátékok és filmek esetében is. Nézzünk meg ezekre is néhány példát!

Jet Set Radio videójáték A *Jet Set Radio* volt a világon az első olyan videójáték amelyen sötét árnyalatú grafikákat mutattak be, túlzott alakokkal, vastag vonalakkal és lapos, élénk színekkel [8]. Ezzel olyan érzetet keltve, mint ha egy képregény vagy egy rajzfilm lenne. A játékban egy görkorcsolyázó karaktert lehet irányítani és graffitiznie kell a megadott helyekre mielőtt lejár az idő. Plusz pontokért lehet mindenféle extrém trükköt is csinálni korcsolyázás közben. Nehezítésként a játékost üldözik gyalogosan, helikopterekkel és tankokkal.

Kamera által homályosan (A Scanner Darkly) című film A *Kamera által homályosan* című filmet digitálisan forgatták, majd interpolált rotoszkóp segítségével animálták [9][10]. Ez egy olyan animációs technika, melyben az eredeti felvételekből képkockáról képkockára filtereztek. Ezzel olyan hatást értek el, hogy azt lehet mondani erre a filmre, hogy animációs film. A film a közeljövőben játszódik. A lakosság nagy része drog fogyasztó. Beépített titkos ügynökökkel és különböző megfigyelési rendszerekkel

próbálnak rendet tartani. Az egyik detektívnek, Frednek kiadják parancsba, hogy álljon rá egy nagyon veszélyes kábítószert terjesztőre, Bob Arctor-ra. A drog hatása az, hogy az emberek személyisége két, egymással ellentétes felére bomlik. Kiderül, hogy Arctor igazából Fred alteregója, igy ez az ügy elég nagy kihívásokkal jár.

Az említett videójátékról és filmről a 2.3. ábrán láthatunk egy-egy képet.

2.3. ábra. Jet Set Radio videójáték és Kamera által homályosan c. film egy-egy jellegzetes képkockája

2.2. A megvalósítás nehézségei

A digitális képfeldolgozás számítógépes algoritmusokat használ a digitális képek készítéséhez. Ez lehetővé teszi, hogy sokkal szélesebb körű algoritmusokat alkalmazzanak a bemeneti adatokra, és ezekkel az algoritmusokkal elkerülhetők az olyan problémák, mint a zaj és a jelek torzulása a feldolgozás során. Mivel a képeket két dimenzióban definiálják, a digitális képfeldolgozást multidimenzionális rendszerek formájában lehet modellezni. Ezekkel a képfeldolgozási algoritmusokkal lehet olyan képeket készíteni, amiket például azok az alkamazások is létrehoznak amiket ebben a fejezetben bemutattam.

2.3. Elvárások a szűrőkkel kapcsolatban

Olyan szűrőket szeretnék létrehozni, amelyek hasonlóképpen működnek, mint azok az applikációk, játékok, filmek amelyeket ebben a fejezetben említettem. Rajzfilm (*cartoon*) jellegű, festmény szerű, valamint ceruza rajz jellegű szűrőket szándékozok létrehozni, amelyeket képekre, videókra lehet alkalmazni. A dolgozatomban ezek számítási idejét tesztelni is fogom.

3. fejezet

Matematikai eszközök

3.1. Zaj szűréshez, elmosáshoz használatos szűrők

A következő szakaszokban annak bemutatására kerül majd sor, hogy a tervezett művészi szűrők létrehozásához milyen matematikai apparátusra van szükség. Ez alapvetően a különféle szűrési eljárások megvalósítását takarja példákkal illusztrálva.

A módszerek digitális képfeldolgozás területén közismertnek tekinthetők. A témakör feldolgozásához Czap László [11] és Kató Zoltán [12] egyetemi kurzusainak anyagai szolgáltak alapul.

3.1.1. Átlagoló szűrő

Az átlagoló szűrő segítségével simítani lehet a képet. A képpontok közelebb kerülnek a környezetük átlagához, azaz a kép "simább" lesz, a szűrt kép intenzitásértékei a kiinulási kép intenzitástartományában maradnak. Csökkenti a zajt, de elmossa az éleket igy homályossá teszi a képet.

3.1. példa. Itt látható a átlagoló szűrésre egy példa, egy képpont 3×3-as környezete:

$$A = \frac{1}{9} \times \begin{bmatrix} 54 & 26 & 32 \\ 17 & 36 & 24 \\ 11 & 23 & 47 \end{bmatrix}.$$

Egyszerű simítási technika, ahol az ablakban lévő intenzitások átlaga az új intenzitás érték:

$$\bar{x} = \frac{54 + 26 + 32 + 17 + 36 + 24 + 11 + 23 + 47}{9} = \frac{270}{9} = 30.$$

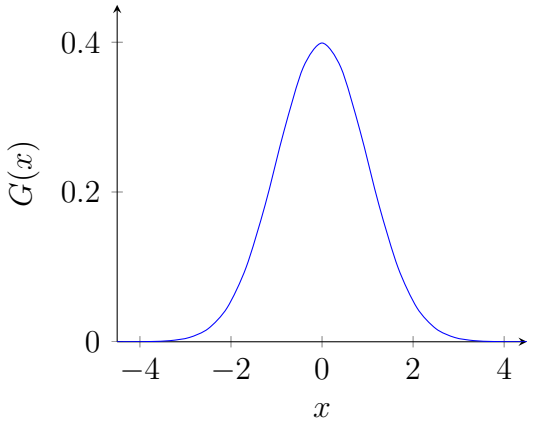
3.1.2. Gauss szűrő

A Gauss szűrő egy bemeneti kép és egy Gauss-kernel konvolúciójával jön létre. Minden egyes képpont értéket a környékének súlyozott átlagaként számolunk úgy, hogy a képpont eredeti értékének a legnagyobb a súlya, míg a távoliak kisebb súlyt kapnak.

Ez olyan elmosódást eredményez, amely jobban védi a széleket, mint más egyenletes elmosódási algoritmusok. Egy Gauss függvény felírása egy dimenzióban az alábbi

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{x^2}{2\sigma^2}}.$$

A Gauss függvény grafikonját a 3.1. ábrán láthatjuk.



3.1. ábra. Az egydimenziós Gauss függvény $\mu = 0$ és $\sigma = 1$ paraméterekkel

Két dimenzióban, két Gauss együttesét kell használni, minden egyes dimenzióban:

$$G(x, y) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{x^2+y^2}{2\sigma^2}},$$

ahol az x a vízszintes tengely eredetétől való távolság, az y a függőleges tengely eredetétől való távolság, a σ pedig a Gauss eloszlás szórása. Két dimenzióban alkalmazva, ez a képlet olyan felületet hoz létre, amelynek körvonalait koncentrikus körök alkotják, a Gauss-eloszlás pedig a középpontból indul.

3.1.3. Medián szűrő

Az $a_1, a_2, \ldots, a_{2n+1} \in \mathbb{R}$ számok mediánja, a nagyság szerint rendezett számsorozat középső, $(n+1)$-edik eleme. Jelölhetjük például a $med\{a_1, a_2, \ldots, a_{2n+1}\}$ formában.

Teljesül rá az alábbiak:

- $\min\{a_i\} \leq med\{a_i\} \leq \max\{a_i\}$,
- $\min\{a_i + c\} = med\{a_i\} + c$,
- $med\{c \cdot a_i\} = c \cdot med\{a_i\}$.

A medián szűrést egy $S \in \mathbb{R}^2$ környezet felett a

$$J(i, j) = med\left\{I(i+u, j+v) \mid (u, v) \in S\right\}$$

formában végezhetjük el.

A medián szűrés eredményét az S környezet mérete (és alakja) határozza meg.

3.2. példa. Itt látható a medián szűrésre egy példa, egy képpont 3×3 méretű környezete:

$$M = \begin{bmatrix} 54 & 25 & 32 \\ 17 & 37 & 22 \\ 11 & 23 & 45 \end{bmatrix}.$$

Nagyság szerint sorba rendezve ezeket az értékeket, úgy hogy 11, 17, 22, 23, 25, 32, 37, 45, 54 akkor a pixel új intenzitása 25 lesz, mivel az a középső érték a sorban.

3.1.4. Kétoldalú szűrő

A kétoldalaú szűrő egy nem lineáris, élvédő és zajcsökkentő simító szűrő a képekhez [13]. Az egyes képpontok intenzitását a közeli pixelek intenzitásának súlyozott átlagával helyettesíti. Ez a súly Gauss eloszláson is alapulhat. Elengedhetetlen, hogy a súlyok nem csak az euklideszi képpontok távolságától, hanem a radiometriai különbségektől is függnek (például tartománykülönbségek, például színintenzitás, mélységi távolság stb.). Ez a kép éles széleit megőrzi. A kétoldalú-szűrő az alábbi alakban írható föl:

$$I^{filtered}(x) = \frac{1}{W_p} \sum_{x_i \in \Omega} I(x_i) f_r(\|I(x_i) - I(x)\|) g_s(\|x_i - x\|),$$
$$W_p = \sum_{x_i \in \Omega} f_r(\|I(x_i) - I(x)\|) g_s(\|x_i - x\|),$$

ahol

- $I^{filtered}$ a filterrel ellátott kép,
- I az eredeti bemeneti kép,
- x a koorditánái a jelenlegi pixeleknek amik a szűrőbe kerülnek,
- ω az ablak közepe x-nek,
- f_r a tartományi kernel az intenzitások közötti különbségek simítására,
- g_s a térbeli kernel a koordináták közötti különbségek simítására.

A W_p súlyt a térbeli közelség és az intenzitás különbség alkalmazásával határoztuk meg. Vegyünk egy pixelt az (i, j) koordinátákon, amelyet a szomszédos képpontok segítségével kell zajcsökkenteni a képen, és az egyik szomszédos pixele a (k, l) helyen található. Ezután a pixelhez (k, l) hozzárendelt súly az (i, j) pixel zajcsökkentéséhez a következőket adja meg:

$$w(i, j, k, l) = \exp\left(-\frac{(i-k)^2 + (j-l)^2}{2\sigma_s^2} - \frac{\|I(i, j) - I(k, l)\|^2}{2\sigma_r^2}\right),$$

ahol σ_s és σ_r a kiegyenlítési paraméterek, és $I(i,j)$ és $I(k,l)$ a pixelek (i,j) és (k,l) intenzitása.

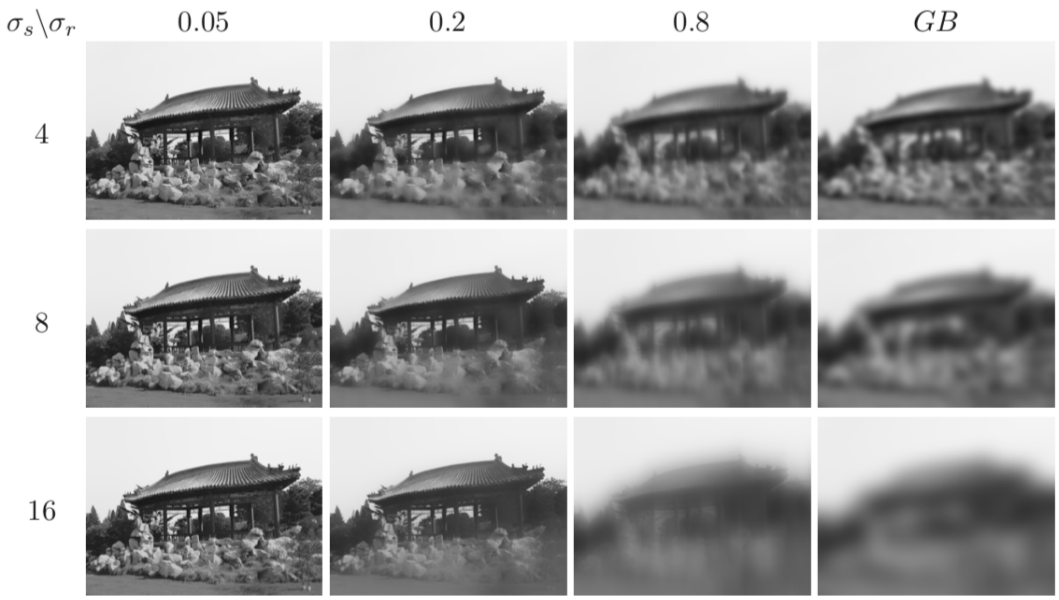
A súlyok kiszámítása után normalizáljuk őket:

$$I_D(i,j) = \frac{\sum_{k,l} I(k,l)w(i,j,k,l)}{\sum_{k,l} w(i,j,k,l)},$$

ahol az I_D a pixel (i,j) zajcsökkentés intenzitása.

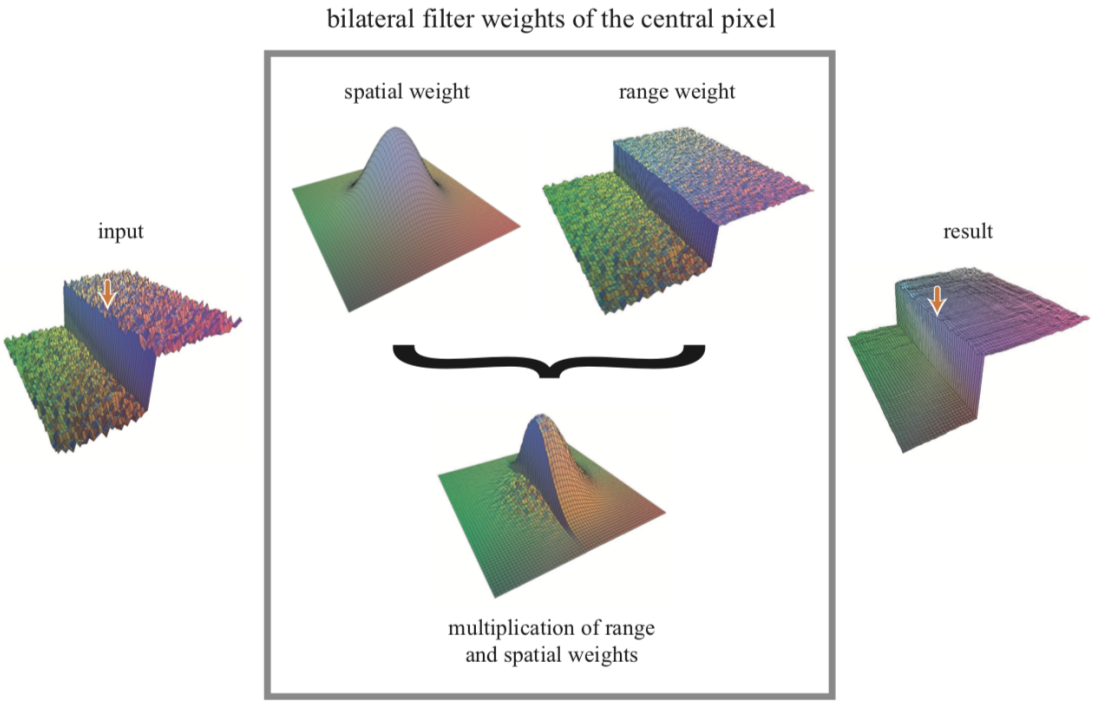
A kétoldalú szűrőt két paraméterrel lehet irányítani: σ_r és σ_s. Ezen paraméterek hatását a 3.2. ábrán láthatjuk.

- Amikor a σ_r paraméter növekszik, a kétoldalas szűrő fokozatosan megközelíti a Gauss konvolúciót, mivel a Gauss-tartomány kiszélesedik és kilapul, ami azt jelenti, hogy a kép intenzitási intervalluma alatt majdnem állandóvá válik.
- Ahogy a σ_s térbeli paraméter nő, a hozzá tartozó nagyobb értékek szerint a kép simább lesz.



3.2. ábra. A σ_r és a σ_s paraméterek hatása, valamint a Gauss elmosással való összehasonlítás.

A 3.3. ábrán láthatunk egy példát a kétoldalú szűrő elmosására egy bemenetként adott képen, amelyen így az élek megőrzésre kerülnek.



3.3. ábra. A kétoldalú szűrő hatása egy bemeneti képen.

3.2. Éldetektálási módszerek

Az éldetektálás számos matematikai módszert foglal magába, amelyek olyan pontok azonosítását célozzák meg egy képen, amelyeknél a kép fényereje élesen megváltozik. A kép egy szeletén vizsgálva az intenzitás-profilt jól azonosíthatóak az objektumok határainak megfelelő változások. Az intenzitás-gradiensből következtethetünk az élek helyére és irányára. Jellegzetes élprofilokat láthatunk a 3.4. ábrán.

A képfüggvény kétváltozós, tehát parciális deriváltakból álló gradiens vektoraink vannak:

$$\nabla f = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right]^T .$$

A hossza a változás nagyságával egyenlő:

$$\|\nabla f\| = \sqrt{G_x^2 + G_y^2} \approx |G_x| + |G_y| .$$

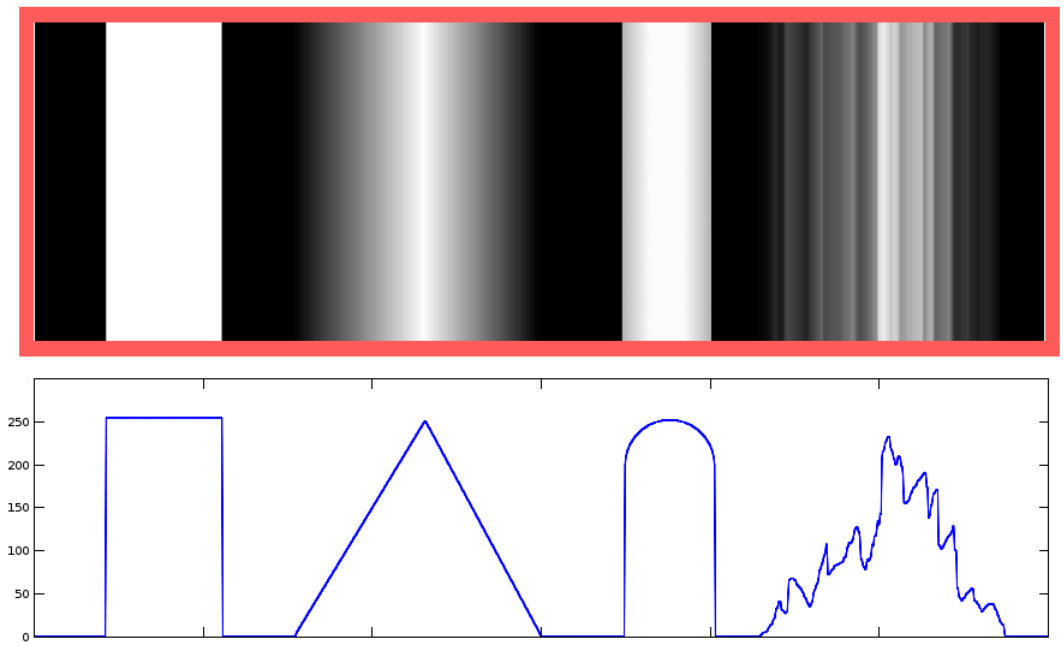
A gradiens a legnagyobb változás irányába mutat:

$$\theta = \tan^{-1}\left(\frac{G_y}{G_x}\right).$$

Digitális képek esetén a pontos, kétváltozós függvényünk nem ismert, ezért a gradiens vektort véges differenciával közelíthetjük:

$$\frac{\partial f}{\partial x} = \lim_{\varepsilon \to 0} \left(\frac{f(x+\varepsilon, y)}{\varepsilon} - \frac{f(x+y)}{\varepsilon} \right) \approx \frac{f(x_{n+1}+y) - f(x_n, y)}{\Delta x}.$$

A gradiens közelítéséhez konvolúciós maszkot is használhatunk.



3.4. ábra. Példa tipikus lépcső, tető, vonal és zajos élprofilokra

3.2.1. Sobel éldetektálás

A Sobel éldetektáló egy gradiens alapú módszer [14]. Ez elsőrendű deriváltakkal működik. A kép első deriváltját külön számítja az x és y tengelyekhez. A deriváltak csak közelítések (mivel a képek nem folytonosak). Ezek közelítéséhez az alábbi kernelt használják a konvolúcióhoz:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \qquad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}.$$

Az első mátrix a vízszintes, a második mátrix a függőleges kernel. A bal oldali kernel megközelíti a deriváltat az x tengely mentén, a jobb oldali pedig az y tengely mentén. Ezen információk felhasználásával számíthatjuk ki az alábbiakat:

- magnitudója vagy "erőssége" az élnek: $\sqrt{G_x^2 + G_y^2}$,
- megközelítő erősség: $\|G_x\| + \|G_y\|$,
- az él iránya: $arctan\left(\frac{G_y}{G_x}\right)$.

3.2.2. Laplace éldetektálás

A Laplace éldetektáló csak egy kernelt használ [14]. Másodfoku deriváltakat számol ki egy lépésben. Itt van néhány elterjedt kernel:

$$G_1 = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \qquad G_2 = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Az első mátrix a laplace operátor, a második mátrix a laplace operátor átlókkal.

Használhatjuk akár csak az egyiket, vagy ha jobb közelítést szeretnénk, létrehozhatunk egy 5x5-ös kernelt (aminek a középpontja 24, és minden más -1).

Egy komoly hátrány azonban van, mivel másodfokú deriváltakkal dolgozunk, a laplace éldetektáló rendkívül érzékeny a zajokra. Általában zajcsökkentés szükséges. Néhány fontos további jellemzője:

- elmosódott élek esetén pontosabb lokalizálást érhetünk el,
- ebben az esetben csak az élek helyét tudjuk meghatározni, az irányát nem,
- az operátor nem érzékeny az elforgatásra.

3.2.3. Canny éldetektálás

Az Canny éldetektálás olyan technika, amely a különböző objektumokból származó hasznos strukturális információkat kivonja, és drasztikusan csökkenti a feldolgozandó adatok mennyiségét. Számos számítógépes képfeldolgozó rendszerben széles körben alkalmazzák. Canny úgy találta, hogy viszonylag hasonlóak a különféle éldetektálás alkalmazására vonatkozó követelmények. Így a követelményeknek megfelelő éldetektáló megoldás számos helyzetben megvalósítható.

Algoritmus:

1. Gauss simítás
2. A kép intenzitás gradiensének megkeresése
3. Nem-maximumok elhagyása
4. Hiszterézis küszöbölés

1. Gauss simítás

Mivel az éldetektálás eredményeit a képzaj is könnyen befolyásolja, elengedhetetlen a zaj kiszűrése, hogy megelőzze a zaj által okozott hamis detektálást. A kép simítása ér-

dekében Gauss szűrőt alkalmazunk. Ez a lépés enyhén simítja a képet, hogy csökkentse a nyilvánvaló zaj hatását a éldetektálásra.

2. A kép intenzitás gradiensének megkeresése

A kép élei különböző irányokban jelennek meg, így a Canny algoritmus négy szűrőt használ a vízszintes, függőleges és átlós élek detektálásához az elmosódott képen. Éldetektáló operátor (például Sobel) az első derivált értékét vízszintes irányban (G_x) és a függőleges irányba (G_y) adja vissza. Ebből meghatározható az él-gradiens és az irány:

$$G = \sqrt{G_x^2 + G_y^2}, \quad \Theta = arctan2\left(\frac{G_y}{G_x}\right).$$

ahol G a hypot függvény segítségével számítható ki, és az arctan2 az arctangens függvény két argumentummal.

Az élirány-szög négy függőleges, vízszintes és két átlós (0 , 45 , 90 és 135 fokok) függőleges szögek egyikére kerekítve van. Az egyes színsávokba eső élek iránya meghatározott szögértékekre van beállítva.

3. Nem-maximumok elhagyása

A nem-maximumok elhagyása a "vékony" élekre alkalmazzák. A gradiens kiszámítása után a gradiens értékből kivont él még mindig homályos. A 3. kritérium vonatkozásában csak egy pontos válasz lehet az élre. Így a nem-maximumok elhagyása segíthet elhagyni a gradiens értékeket (0-ra állítva), kivéve a lokális maximumokat, amelyek a legerősebb intenzitásérték-változást jelzik. Az algoritmus a következő, minden pixelre a gradiens képen:

- Hasonlítsa össze az aktuális képpont él-erősségét a pixel él-erősségével, pozitív és negatív gradiens irányba.
- Ha az aktuális képpont él-erőssége a legnagyobb az azonos irányú maszk más képpontjaihoz képest, az érték megmarad. Ellenkező esetben az érték elhagyásra kerül.

Bizonyos implementációkban az algoritmus a folyamatos gradiens irányokat különálló diszkrét irányokba sorolja, majd egy 3×3 szűrőt rak az előző lépés kimenetre. Minden pixelben elhagyja a középső képpont él-erősségét (az értékét 0-ra állítva), ha nagysága nem nagyobb, mint a két szomszéd nagysága a gradiens irányba.

4. Hiszterézis küszöbölés

Eddig az erős élű képpontokat minden bizonnyal be kell vonni a végső élbe, mivel ezek a kép valódi éleiből származnak.Vannak azonban viták a gyenge élű képpontokról, mivel ezek a pixelek kiválaszthatók az igazi élről, vagy a zaj/színváltozatokról. Pontos

eredmény elérése érdekében az utóbbi okok által okozott gyenge éleket el kell távolítani. Általában gyenge él pixeleket okoznak, az igazi élek amikor erős élű pixelekhez kapcsolódnak, amíg a zajok nem kapcsolódnak hozzá. Az élkapcsolat nyomon követéséhez a blob elemzést kell végre hajtani, ami egy gyenge élű pixel és 8-as kapcsolatú szomszédos pixelek figyelembevétele. Mindaddig, amíg van egy erős élű pixel, amely részt vesz a blob elemzésben, a gyenge élű pontot lehet azonosítani, amit meg is kell őrizni.

3.3. Szegmentálás

Számítógépes látásmódban a képszegmentálás a digitális kép több szegmensbe való felosztása. A szegmentálás célja egyszerűsíteni és/vagy megváltoztatni a kép reprezentációját ezzel könnyebbé téve a kép elemzését. A kép szegmentálását általában objektumok és határok megtalálásához használják. Pontosabban, a képszegmentálás egy címke hozzárendelését jelenti a kép minden képpontjához úgy, hogy az azonos címkével rendelkező pixelek bizonyos tulajdonságokkal rendelkeznek. A képszegmentáció eredménye olyan szegmensek csoportja, amelyek együttesen fedik le a teljes képet vagy a képből kinyert kontúrt. Valamennyi régióban lévő pixelek hasonlóak bizonyos jellemzők vagy számított tulajdonságok, például szín, intenzitás vagy textúra tekintetében. A szomszédos régiók szignifikánsan különböznek az azonos jellegzetességek tekintetében.

3.3.1. Mean shift algoritmus

Mean shift egy eljárás a maximális értékek azonosítására, ahol a sűrűségfüggvény módjait a funkciótól vett diszkrét adat szolgáltatja. Ez egy iteratív módszer, az x kezdeti becslésével kezdünk. Adjuk meg a $K(x_i - x)$ kernel függvényt. Ez a függvény határozza meg a közeli pontok súlyát az átlag újraértékeléséhez. Általában egy Gauss kernelt használunk az aktuális becsléshez képest, $K(x_i - x) = e^{-c||x_i - x||^2}$. A K által meghatározott ablak sűrűségének súlyozott átlaga:

$$m(x) = \frac{\sum_{x_i \in N(x)} K(x_i - x) x_i}{\sum_{x_i \in N(x)} K(x_i - x)},$$

ahol $N(x)$ az x szomszédsága, olyan pontok halmaza, amelyekhez $K(x_i) \neq 0$.

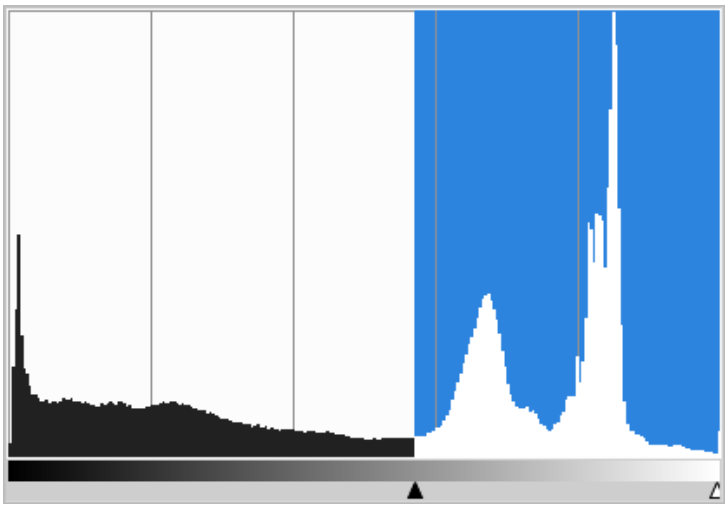
Az $m(x) - x$ különbséget az Fukunaga és a Hostetler átlagos eltolódásának nevezik. Az mean shift algoritmus most beállítja az $x \leftarrow m(x)$ értéket, és megismétli a becslést, amíg $m(x)$-ig konvergál.

3.4. Küszöbölés

A küszöbérték a képszegmentálás legegyszerűbb módja. A szürkeárnyalatos képből küszöbérték használatával bináris képek készíthető. Tételezzük fel, hogy az objektum és

a háttér eltérő intenzitású, vagyis nagy a kontraszt. Az objektum és a háttér önmagában homogén intenzitású. A küszöbölést így a hisztogram alapján választott megfelelő értékkel el tudjuk végezni (3.5. ábra).

Zajos kép esetén nehéz kielégítő köszöbértéket meghatározni, tehát a szegmentálás inhomogén lesz, erre a zajszűrés jelenthet megoldást.



3.5. ábra. Példa a hisztogram alapján egy megfelelő küszöbérték megválasztására

3.3. példa. Sötét objektum világos háttérben:
$f(x, y)$- bemeneti kép;
$t(x, y)$- szegmentált kép;
T=küszöbérték

$f(x, y) \leq T$, akkor $t(x, y) = 1$, azaz objektum.
$f(x, y) > T$, akkor $t(x, y) = 0$, azaz háttér.

Léteznek adaptív eljárások, melyek az input képhez automatikusan választják ki az optimális küszöbértéket. Ebből vannak globális és lokális küszöbértéket meghatározó algoritmusok. A globális köszöbölésre alkalmas algoritmusok hisztogramból klaszterezéssel számított értékekel dolgoznak. Ilyen például az Isodata valamint az Otsu algoritmus. A lokális küszöbölés változó értékekkel dolgozik, például egyenletlen megvilágítás. Amennyiben az objektum és a háttér kontrasztja globális küszöbölés után, lokálisan még mindig nagy, akkor alkalmazzuk a lokális küszöbölést, például Niblack algoritmust.

3.4.1. Izodata algoritmus (Yanni)

Jól használható, ha az előtér és a háttér körülbelül ugyanannyi képpontból áll. Az algoritmus az alábbi lépésekből áll.

1. Inicializálás: a hisztogramot kétrészre osztjuk, célszerűen a felezőponton: T_0

2. Kiszámítjuk az objektum valamint a háttér intenzitásának a középértékét: M_i, m_i
3. Az új küszöbérték a két középérték átlaga: $T_i = (M_i + m_i)/2$
4. Vége ha a küszöbérték már nem változik: $T_k + 1 = T_k$

3.4.2. Otsu algoritmus

A bemeneti kép L szürkeárnyalatot tartalmaz, a normalizált hisztogram minden x szürkeértékéhez megadja az előfordulási gyakoriságát, vagyis a valószínűségét: p_x. Keressük azt a T küszöbszámot, amely maximalizálja az objektum-háttér közötti varianciát.

Az előtér/háttér pixelek gyakorisága és középértéke:

Háttér valószínűsége:

$$B(T) = \sum_{x=1}^{T} p_x$$

Objektum valószínűsége:

$$1 - B(T) = \sum_{x=T+1}^{L} p_x$$

Legyen $m(T) \equiv \sum_{x=1}^{T} xp_x$, a teljes kép középértéke: $\mu \equiv m(L) = \sum_{x=1}^{L} xp_x$,

Háttér középértéke:

$$\mu_B = \frac{m(T)}{B(T)}$$

Objektum középértéke:

$$\mu_O = \frac{\mu - m(T)}{1 - B(T)}$$

Az előtér/háttér pixelek szórása:

$$\sigma_B^2 = \frac{1}{B(T)} \sum_{x=1}^{T} (x - \mu_B)^2 p_x,$$

$$\sigma_O^2 = \frac{1}{1 - B(T)} \sum_{x=T+1}^{L} (x - \mu_O)^2 p_x$$

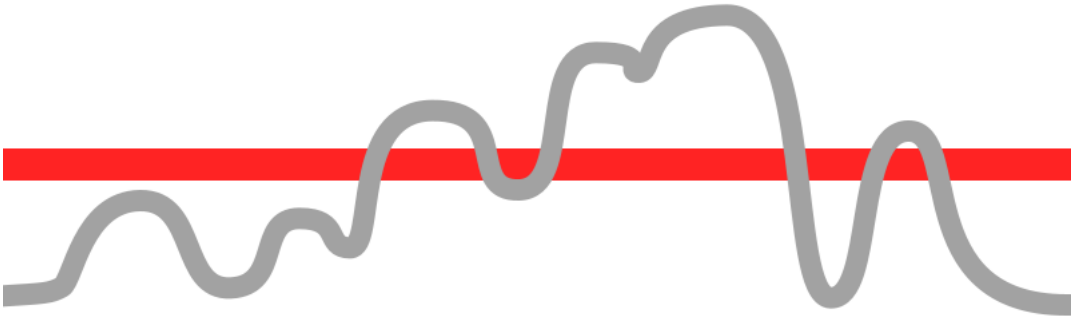
A teljes kép szórása:

$$\sigma^2 = \sum_{x=1}^{T} (x - \mu)^2 p_x + \sum_{x=T+1}^{L} (x - \mu)^2 p_x = \cdots =$$

$$= B(T)\sigma_B^2 + (1 - B(T))\sigma_O^2 + (\mu_B - \mu)^2 B(T) + (\mu_O - \mu)^2 (1 - B(T)) \equiv \sigma_W^2(T) + \sigma_C^2(T)$$

ahol, a $B(T)\sigma_B^2 + (1 - B(T))\sigma_O^2 = \sigma_W^2(T)$, osztályon belüli varianciától függ és a $(\mu_B - \mu)^2 B(T) + (\mu_O - \mu)^2 (1 - B(T)) = \sigma_C^2(T)$, osztályokközötti varianciától függ. A σ^2 konstans, és T-t úgy kell beállítani, hogy $\sigma_C^2(T)$ a lehető legnagyobb legyen:

$$\sigma_C^2(T) = \cdots = \frac{(\mu(T) - \mu B(T))^2}{B(T)(1 - B(T))}$$



3.6. ábra. Intenzitásértékek küszöbölése globális küszöbérték mellett

3.7. ábra. Intenzitásértékek küszöbölése lokális/adaptív küszöbértékkel

A hisztogram elejéről kezdve nézzük meg minden szürkeértéket, mint lehetséges küszöböt. Kiszámoljuk a $\sigma_C^2(T)$ értéket a $\mu(T)$ és $B(T)$ segítségével, mindaddig növeljök a T értékét, amíg $\sigma_C^2(T)$ kövekszik. Ez az algoritmus feltételezi, hogy $\sigma_C^2(T)$-nek egy maximuma van.

3.4.3. Niblack algoritmus

Egyetlen küszöb nem elegendő az objektum és a háttér szétválasztásához. Ezért változó köszöbérték ($T(i,j)$) kell, amely követi az intenzitás változásokat:

$$T(i,j) = \mu(i,j) + k\sigma(i,j)$$

Az (i,j) adott környezetében, $\mu(i,j)$ a középértéket jelenti, $\sigma(i,j)$ pedig a szórást. A k egy súly, ami megmutatja mennyire vegyük figyelembe a szórást.

Ha $k < 0$, akkor sötét az objektum, Ha $k > 0$, akkor világos az objektum.

A globális és a lokálisan változó küszöbértékre láthatunk egy-egy szemléltetést a 3.6. és a 3.7. ábrákon.

4. fejezet

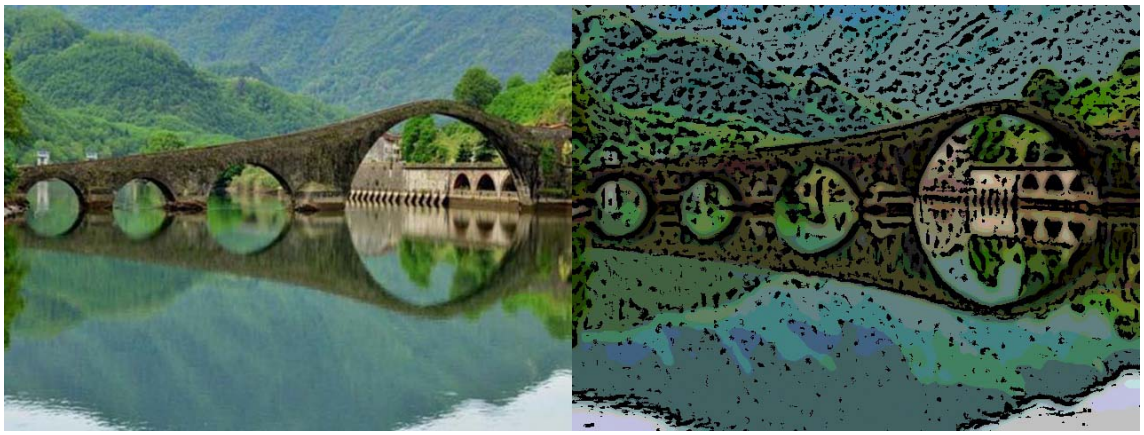
Szűrő algoritmusok

Ebben a fejezetben azokat a filtereket részletezem, amelyeket C++ programozási nyelven készítettem el. Többségükben rajzfilm, valamint festmény jellegűek. A szűrőket nem csak képeken teszteltem, hanem videókon is és valós időben is a számítógépem kamerájával. (Az utóbbiak eredményeit a 6. fejezet részletezi.)

4.1. Cartoon-style filter

Az első filter egy rajzfilm jellegű (*cartoon-style*) filter, amely az éleket kiemeli és a színeket elmossa. A szűrőhöz az ötletet Michael Beyeler hasonló szűrője adta, melyet Python nyelven készített el [15].

A műveletekhez Gauss-piramist, kétoldalú szűrést, medián szűrést, illetve adaptív küszöbölést használtam. A szűrés eredményére egy példát a 4.1. ábrán láthatunk.



4.1. ábra. Bal oldalon az eredeti-, jobb oldalon pedig a filterrel ellátott kép látható (forrás: *http://www.erdekesvilag.hu/elkepeszto-tajkepek/*)

4.1.1. Gauss-piramis és kétoldalú szűrő

Első lépésben az eredeti képet lekicsinyítettem Gauss-piramis segítségével a kép méretének negyedére, rátettem egy kétoldalú szűrőt, majd vissza nagyítottam az eredeti méretre. A Gauss-piramis a kép kicsinyítése előtt Gauss-simítás segítségével súlyoz. A kétoldalú szűrő egy nem lineáris, élvédő és zajcsökkentő simító szűrő. Ezen lépés eredménye látható a 4.2. ábrán.

4.2. ábra. A Gauss-piramisban kicsinyítés és nagyítás valamint a kétoldalú szűrő eredménye

4.1.2. Színek konvertálása, medián szűrő

Ebben a lépésben először az előzőleg kapott elmosott, színes képet átkonvertáltam szürkeárnyalatos képpé. Itt ki szeretnék térni magára a színes kép szürkeárnyalatossá konvertálására. Több féle megoldás létezik, ebből az elterjedtebbeket említeném. Az egyik az RGB komponensek átlagolása, de ez az emberi szem számára pontatlan, mivel nem egyformán érzékelünk minden színt. A következő lehetőség, amit az OpenCV-beépített színkonvertálása is használ az, hogy a komponensek súlyozott átlagát veszi, de nem egyenlő súlyértékekkel. Ennek a speciális esete, amely a HSV színrendszer *value* értékét számítja. Ebben az esetben az OpenCV-s beépített kovertálást használtam a következő súlyokkal:

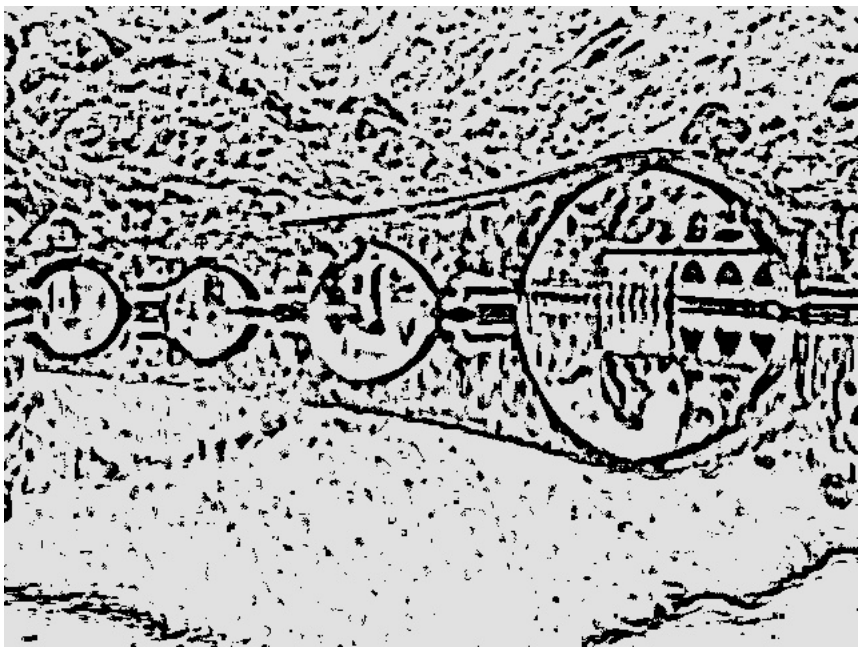
$$\text{RGB to Gray} = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B.$$

Ennek eredményeként kapott képen medián szűrést hajtottam végre, amit ismét zajcsökkentés miatt alkalmaztam. Így a kép már teljesen el van mosva, egyre jobban rajzfilmszerű hatása van. Ennek az eredménye a 4.3. ábrán látható.

4.3. ábra. Szürkére átalakított kép medián szűrővel

4.1.3. Adaptív küszöbölés az élek kiemelésére

Az adaptív küszöbérték általában szürkeárnyalatos vagy színes képet kap bemenetként, és a legegyszerűbb megvalósításban bináris képet eredményez. A kép minden egyes képpontjára egy küszöbértéket kell kiszámítani. Ez a küszöbérték lehet például egy kernelen belüli intenzitások átlaga, vagy egy, a hisztogram alapján becsült érték. Amennyiben a képpontérték a küszöbérték alatt van, akkor a háttérértékre van állítva, ellenkező esetben az előtérbe kerül.



4.4. ábra. Az adaptív küszöbölés eredménye

4.1.4. A szűrőkkel és a küszöböléssel előállított képek egyesítése

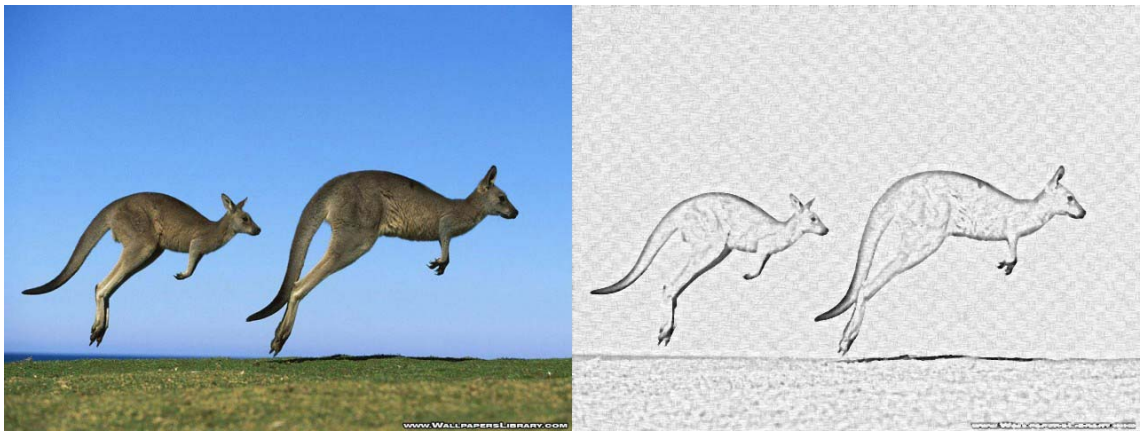
Az adaptív küszöböléssel elkészült maszkot át kell konvertálni színes képpé, hogy egyesíteni tudjuk az "elmosott" képpel amit az első két lépésben hoztunk létre. Ennek az elvégzését követően a 4.5. ábrán látható eredményt kaptuk.

4.5. ábra. Cartoon-style szűrő

4.2. Pencil sketch filter

Megpróbáltam egy olyan algoritmust létrehozni, ami egy ceruza rajzot imitál. A szakirodalomban találunk hasonló szűrési eljárásokat, mivel egy gyakran előforduló hatásról van szó [16].

A feldolgozáshoz szűrkeárnyalatosra konvertáltam a képet, medián szűrőt, Gauss szűrőt és egyéb képfeldolgozási műveleteket használtam, valamint egy "vásznat" is ráraktam, hogy még jobban olyan érzete legyen a képnek mint ha rajzolták volna (4.6. ábra).



4.6. ábra. Bal oldalon az eredeti kép látható, jobb oldalon a filterrel ellátott kép (forrás: *https://amazing.zone/kangaroos/they-only-can-go-forward*)

4.2.1. Medián szűrő

A kép szűrkeárnyalatos konvertálása után, végrehajtottam egy medián szűrést a kisebb, pont szerű zajok eltávolítása céljából (4.7. ábra). A kernel méretétől függően már ez ad egy enyhe elmosást is a képhez, amely a végeredmény szempontjából előnyös.

4.7. ábra. Szürkére átalakított kép medián szűrővel

4.2.2. Gauss szűrő

Ezt a simítási teknikát, a képzaj és a részletesség csökkentése érdekében használtam. Olyan sima elmosódást eredményez a képen, mint ha az nem lenne fókuszban (4.8.

ábra). A medián szűrőhöz képest lényeges különbség, hogy ezzel új árnyalatok is megjelennek a képen, a kontraszt összességében csökken.

4.8. ábra. Gauss szűrő használata

4.2.3. Az előző két lépés elosztása

Az előző két szűrőt elosztottam egymással, így már tényleg majdnem olyan képet kaptam ami már ceruza rajz szerű. Úgy gondoltam még azért ráfér némi javítás, így még további műveleteket hajtottam végre rajta.

4.2.4. Kontraszt széthúzása

Azt figyeltem meg, hogy ha így hagyom a "Pencil sketch" szűrőt, akkor némely kép eléggé kontraszt szegény marad. Emiatt célszerűnek tünt belerakni egy kontraszt széthúzást, a szebb végeredmény érdekében. A széthúzás (vagy másnéven *normalizáció*) egy egyszerű képjavító eljárás, amely megpróbálja javítani a kép kontrasztját azáltal, hogy "kiterjeszti" a benne lévő intenzitásértékek tartományát a kívánt értéktartományra. Ezt a műveletet követően a példaként szereplő képre a 4.10. ábrán látható eredményt kaptam.

4.2.5. Vászon hozzáadása

A kontraszt széthúzása után, már egészen olyan hatása volt a képnek, mint ha egy ceruza rajz lenne. Ráraktam még egy "vászont" ami személyes véleményem szerint mégjobban segíti azt az érzetet hogy ez egy ceruzarajz. Először a vászonnak a színét át kellett konvertálnom színesből szűrkeárnyalatossá, hogy használható legyen a

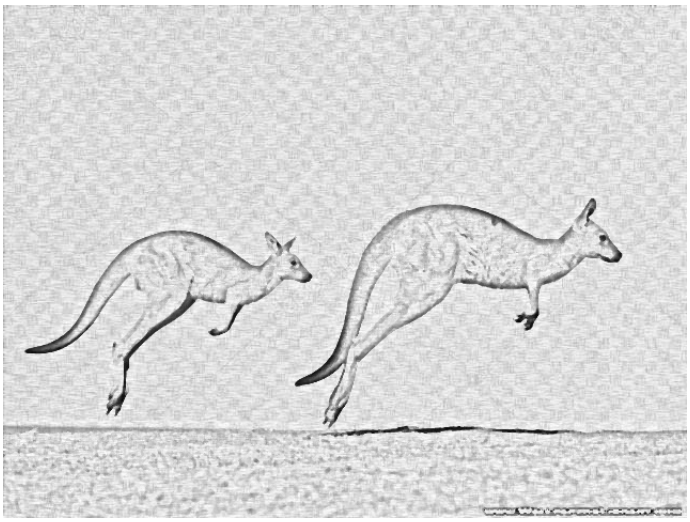
4.9. ábra. Medián szűrés és a Gauss szűrés hányadosa

4.10. ábra. Kontraszt széthúzás

kontraszt széthúzott képhez (4.11. ábra). Ezek után a vászont összeszoroztam az eddig eredményül kapott képpel, így jött létre a "Pencil sketch filter".

A 4.12. ábrán láthatjuk a végeredményként kapott képet.

4.11. ábra. Vászon színének konvertálása



4.12. ábra. A *Pencil sketch filter* eredménye

4.3. Cartoon filter

Megpróbáltam más technikával is létrehozni egy cartoon hatású képet, mivel a fellelhető irodalmakban is számos, különféle megoldást találni [17].

Ahol medián szűrőt, Laplace éldetektálást valamit küszöbölést használtam (4.13. ábra).

4.3.1. Medián szűrő

Mint az eddigi saját filtereknél, itt is előfeldolgozásként elvégeztem szürkeárnyalatossá konvertálást valamint egy medián szűrést (4.14. ábra).

4.3.2. Laplace éldetektálás

Ennél a filternél a Laplace éldetektálást választottam. Ezzel olyan élmaszkot tudtam készíteni, amely kicsit hasonló a ceruza rajzhoz. Vékony kontúrként emeli ki a képben

4.13. ábra. Bal oldalon az eredeti kép látható, jobb oldalon a filterrel ellátott kép (forrás: *http://wallpaper-gallery.net/single/wallpaper-full-hd-nature/wallpaper-full-hd-nature-1.html*)

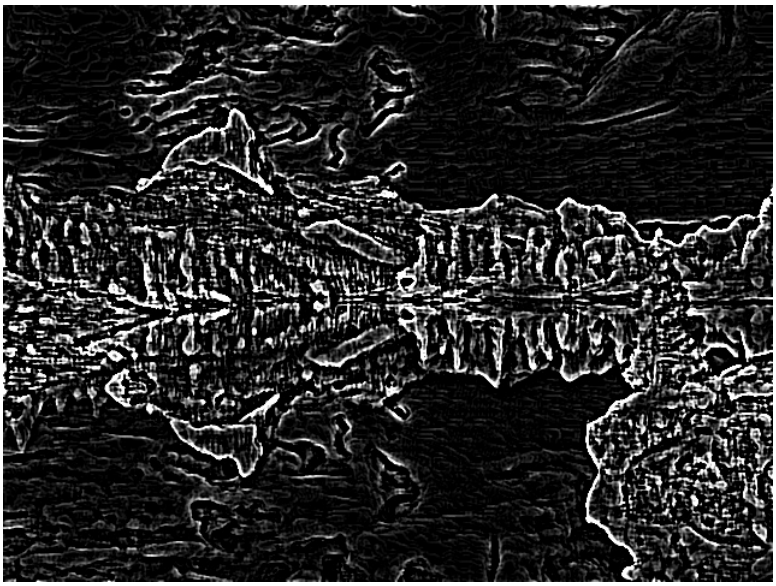
4.14. ábra. Szürkére átalakított kép medián szűrővel

található éleket (4.15. ábra).

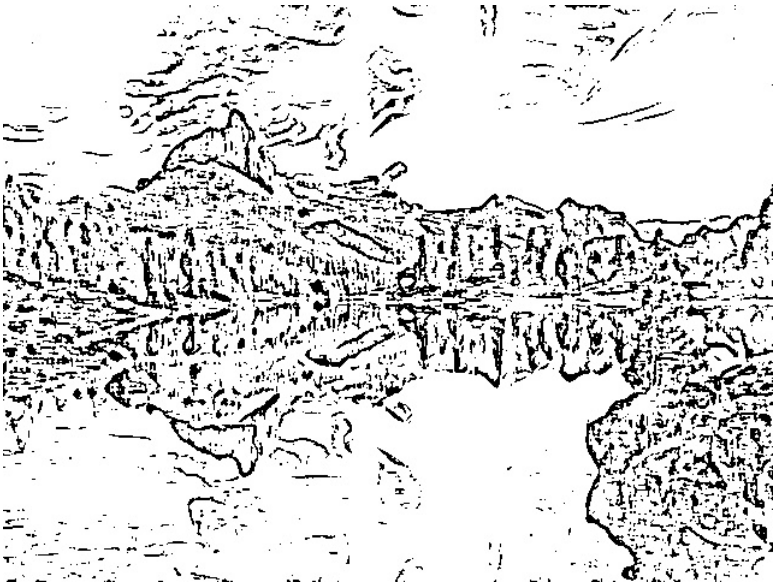
4.3.3. Küszöbölés

Az éldetektálás után, alkalmaztam egy küszöbölést amivel létrejött egy maszk, amit összeraktam az eredeti képpel (4.16. ábra).

Így készült el végeredményként a *Cartoon filter*, melynek az eredménye a 4.17. ábrán látható.



4.15. ábra. Laplace éldetektálás



4.16. ábra. Küszöbölés

4.4. Aquarelle-style filter

Az utolsó saját szűrő amit készítettem az egy egyszerű vízfesték (*aquarell*) hatású szűrő. Ez a szűrő teljesen más technikával készült mint az eddigiek. Nem használtam éldetektálást illetve küszöbölést sem. Az eredeti képpel dolgoztam végig, nem maszkoltam. A szűrés két lépésben megoldhatónak bizonyult. Elég egy átlagoló szűrés és egy mean shift szegmentálás egymást követően alkalmazni (4.18. ábra).

4.17. ábra. Cartoon filter

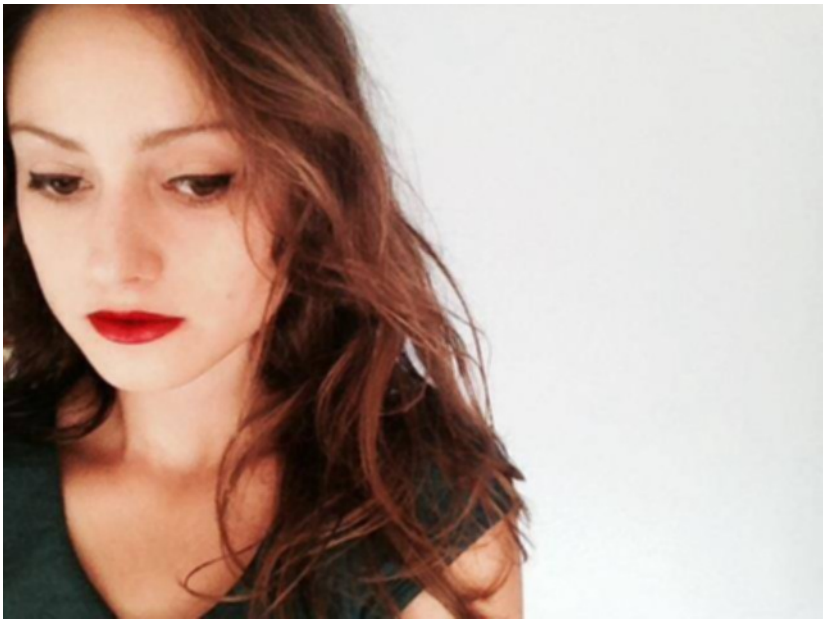
4.18. ábra. Aquarelle-style filter (forrás: *https://marieclaire.hu/kultura/2015/11/24/nem-mindig-az-a-cel-hogy-a-nezo-jol-erezze-magat-interju-kokai-tundevel/*)

4.4.1. Átlagoló szűrő

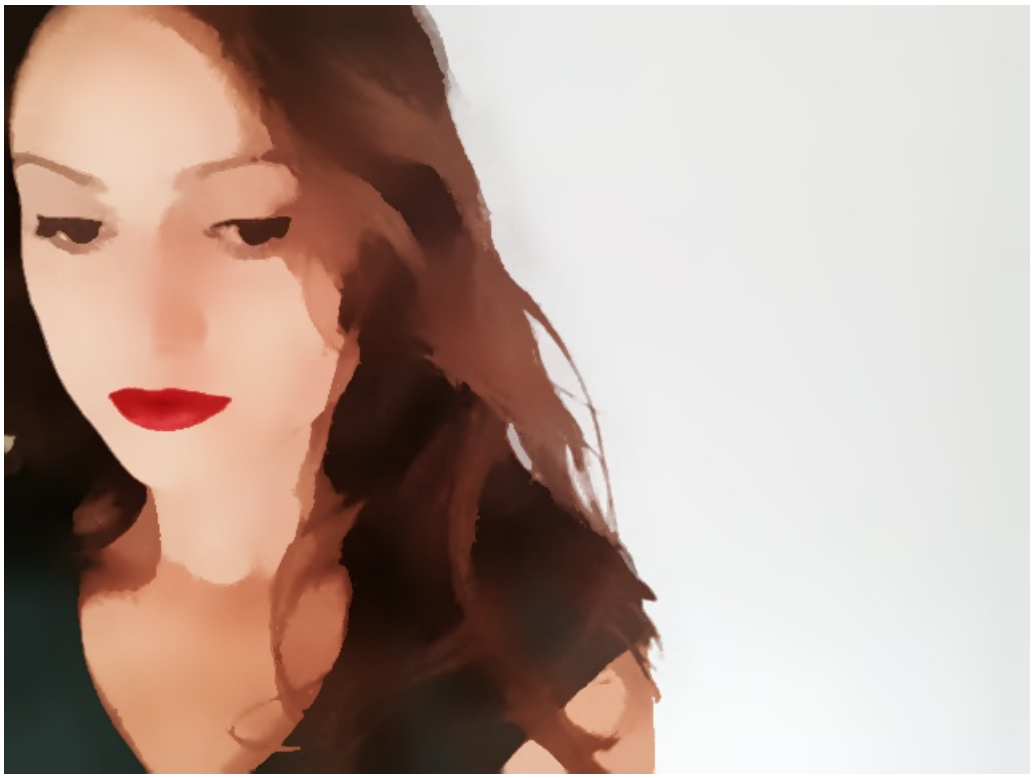
Átlagoló szűrővel elmostam az eredeti színes képet, hogy az apróbb zajokat kiszűrjem (4.19. ábra).

4.4.2. Mean shift szegmentálás

Minden egyes adatpontnál a mean shift meghatároz egy ablakot, és kiszámítja az adatpont átlagát. Ezután az ablak közepét az átlag felé tolja, és megismétli az algoritmust, amíg konvergens. A mean shift egy nemparametrikus iteratív algoritmus vagy egy nemparametrikus sűrűségi gradiens becslés egy általánosított kernel megközelítés alkalmazásával (4.20. ábra).



4.19. ábra. Átlagoló szűrő



4.20. ábra. Aquarelle-style filter, mean shift szegmentálás eredménye

5. fejezet

Implementáció C++ és OpenCV segítségével

5.1. OpenCV bemutatása

Az OpenCV (Open Source Computer Vision Library) egy nyílt forráskódú számítógépes látás és gépi tanulási szoftverkönyvtár [18]. Az OpenCV-t azért hozták létre, hogy közös infrastruktúrát biztosítson a számítógépes megjelenítési alkalmazások számára, és felgyorsítsa a gépi érzékelés használatát a kereskedelmi termékekben. BSD licenc alatt került kiadásra, ezért ingyenes mind tudományos, mind kereskedelmi célokra. C++, Python és Java interfészekkel rendelkezik, és támogatja a Windows, Linux, Mac OS, iOS és Android rendszereket. Az OpenCV-et a számítási hatékonyságra tervezték, és nagy hangsúlyt fektetett a valós idejű alkalmazásokra. Az optimalizált C/C++-ban írt, a könyvtár kihasználhatja a többmagos feldolgozást is. Az OpenCL használatával kihasználhatja az alapul szolgáló heterogén számítási platform hardveres gyorsítását.

A könyvtár több mint 2500 optimalizált algoritmussal rendelkezik, amely magába foglalja mind a klasszikus, mind a legmodernebb számítógépes látásmódot és a gépi tanulási algoritmusokat. Ezek az algoritmusok felismerhetik az arcokat, felismerhetik az objektumokat, osztályozhatják az emberi cselekvéseket a videókban, nyomon követhetik a mozgásokat, követhetik a mozgó objektumokat, eltávolíthatja a vörös szemeket a vakuval készített képekből, követheti a szemmozgásokat, felismerheti a tájat stb.

5.2. Beépített művészi jellegű OpenCV-s szűrők

Az OpenCV könyvtár rendelkezik saját bepített, nem fotorealisztikus szűrőkkel, amelyeknél csak a bemeneti képet, valamint néhány paramétert kell megadnunk a függvénynek és ő elkészíti nekünk a filterezett képet. Négy ilyen beépített függvény található az OpenCV-ben, Edge Preserving filter, Detail Enhancing filter, Pencil sketch filter és a Stylization filter. Ezek C++ implementációját szeretném bemutatni.

5.2.1. Edge Preserving filter

Az első ilyen szűrő az *Edge Preversing filter*, ami az éleket megörzi de a hátteret elmossa (5.1. ábra). Ennek a paraméterezése az alábbi

```
edgePreservingFilter(Mat src, Mat dst, int flags=1,
                     float sigma_s=60, float sigma_r=0.4f);
```

szignatúrájú, ahol

- `src`: a bemeneti kép, amit szeretnénk átalakítani,
- `dst`: a filterrel ellátot kép,
- `flags`: maga az élkiemelő filter melynek két paramétere lehet, `RECURS_FILTER` (rekurív szűrő) aminek az értéke 1 vagy `NORMCONV_FILTER` (normalizált konvolúció) aminek az értéke 2. A futási sebességtől függ melyik paramétert alkalmazzuk, ha gyorsabb sebességet akarunk elérni akkor a rekurzív filtert érdemes használni, ha viszont nem számít a sebesség akkor a normalizált konvolúciót, mert az szebben kiemeli az éleket.
- `sigma_s`: egy skálaérték 0 és 200 között (*sigma spatial*), a simítás mértékét határozzuk meg vele,
- `sigma_r`: egy skálaérték 0 és 1 között (*sigma range*), azt szabályozza, hogy a szomszédságon belül a különböző színek milyen mértékben átlagolódjanak.

5.1. ábra. Eredeti kép, Rekurzív filter, Normalizált konvolúció
(forrás: *https://sq.wikipedia.org/wiki/Papagalli*)

5.2.2. Detail Enhancing filter

A második ilyen szűrő a *Detail Enhancing filter*, ami a képet élesebbé teszi (5.2. ábra).

```
detailEnhance(Mat src, Mat dst, float sigma_s=10, float sigma_r=0.15f);
```

A paraméterek megegyeznek az Edge Preserving filter paramétereivel.

5.2. ábra. Eredeti kép, Detail Enhancing filter eredménye

5.2.3. Pencil sketch filter

Ez a szűrő ceruza rajzot eredményez. Két féle képet is visszaad: egy színes képet és egy szürkeárnyalatosat (5.3. ábra).

```
pencilSketch(Mat src, Mat dst_gray, Mat dst_color, float sigma_s=60,
             float sigma_r=0.07f, float shade_factor=0.02f);
```

A paraméterek megegyeznek az Edge Preserving szűrőével, de bővül az alábbi paraméterrel:

- `shade_factor`: ami egy 0 és 0,1 közötti érték, a kimeneti képintenzitás skálázása. Minél magasabb az érték, annál fényesebb az eredmény.

5.3. ábra. Eredeti kép, Pencil sketch filter színes, Pencil sketch filter szűrkeárnyalatos

5.2.4. Stylization filter

A *Stylization filter* olyan kimeneti képet eredményez, aminek olyan hatása van mint ha vízfestékkel készítették volna (5.4. ábra).

```
stylization(Mat src, Mat dst, float sigma_s=60, float sigma_r=0.45f);
```

A paraméterek megegyeznek az Edge Preserving filter paramétereivel.

5.4. ábra. Eredeti kép, Stylization filter eredménye

5.3. Saját szűrők implementációja

Ebben a fejezetben szeretném bemutatni a saját szűrőknek a C++ implementációját és a paramétereket amiket használtam hozzájuk. Ezeket itt már képekkel nem fogom illusztrálni, mert azok a 4. fejezetben megtalálhatóak.

Elsőként bemutatnám a kép betöltésének implementációját, amit minden szűrőn használtam.

```
Mat img = imread("image.jpg"); // bemeneti kep
Mat dst; // kimeneti kep
// Ha valami nem stimmel a kep betoltesnel
if (img.empty()) {
    cout << "Error loading the image" << endl;
    return -1;
}
```

Az ablakok létrehozását a következőképpen oldottam meg:

```
namedWindow("Original image", 1);
namedWindow("Image with filter", 1);
```

Ezután következik az a rész, amit a következő alfejezetekben fogok részletezni, hogy tulajdonképpen milyen lépésekből épül fel a filter. Ezek után a kép megjelenítése:

```
imshow("Original imagel", img);
imshow("Image with filter", dst);
```

Végezetül a `waitKey` funkció következik, amely a megadott milliszekundumban megjeleníti a képet. Mivel itt nem mozgó képről beszélünk, így az értéke 0.

```
waitKey(0);
```

5.3.1. Cartoon-style filter

Ennek megvalósításához először a Gauss piramisban lefelé lépünk kettőt.

```
for (int i = 0; i < 2; i++) {
    pyrDown(img, img);
}
```

A `pyrDown` függvény paraméterei:

- `img`: a bemeneti színes kép,
- `img`: a kimeneti kicsinyített színes kép.

Egymás után hétszer hajtjuk végre a kétoldalú szűrőt az alábbi kód segítségével.

```
Mat img_res;
int d = 9;
double sigmaColor = 9.0;
double sigmaSpace = 7.0;
for (int i = 0; i < 7; i++) {
    bilateralFilter(img, img_res, d, sigmaColor, sigmaSpace);
}
```

Ebben az

- `img`: a bemeneti színes kép,
- `img_res`: a kimeneti kép kétoldalú szűrővel,
- `d`: a szűrés során használt minden egyes pixel szomszédság átmérője. Ha nem pozitív, akkor kiszámítható a `sigmaSpace` értékéből,
- `sigmaColor`: szűri a szigmát a színtérben,
- `sigmaSpace`: szűri a szigmát a koordinátatérben.

Gauss piramisban felfelé lépünk kettőt, hogy visszanyerjük az eredeti kép méretet.

```
for (int i = 0; i < 2; i++) {
    pyrUp(img_res, img_res);
}
```

amelynek a `pyrUp` függvényében az

- `img_res`: a bemeneti kép kétoldalú szűrővel,
- `img_res`: a kimeneti nagyított színes kép.

Egy képet a

```
cvtColor(img, srcGray, CV_BGR2GRAY);
```

utasítással konvertálhatunk szürkeárnyalatossá, amelyben

- `img`: a bemeneti színes kép,
- `srcGray`: a kimeneti szürkeárnyalatos kép,
- `CV_BGR2GRAY`: színes kép szürkeárnyalatosra alakítására használatos paraméter.

A következő lépésben a medián szűrőt implementáltam az

```
int kernel_size = 7;
medianBlur(srcGray, median, kernel_size);
```

formában, ahol

- `srcGray`: a bemeneti szürkeárnyalatos kép,
- `median`: a kimeneti medián szűrővel ellátott kép,
- `kernel_size`: a kernel mérete, itt ebben az esetben egy 7×7-es mátrix. Ami már nem csak a zaj kiszűrését eredményezi, hanem már cartoon jellegűre mossa a képet.

Az adaptív küszöbölés implementációja az alábbi.

```
Mat img_edge;
double maxValue = 225;
int blockSize = 9;
double C = 2;
adaptiveThreshold(median,img_edge, maxValue, ADAPTIVE_THRESH_MEAN_C,
                  THRESH_BINARY, blockSize, C);
```

amelyben

- `median`: a bemeneti medián szűrővel ellátott kép,
- `img_edge`: a kimeneti éldetektálás eredménye,
- `maxValue`: nem zérus érték hozzárendelve azokhoz a képpontokhoz, amelyekhez a feltétel teljesül.

- `ADAPTIVE_THRESH_MEAN_C`: adaptiv küszöbölés algoritmus,
- `THRESH_BINARY`: a küszöbérték típusának `THRESH_BINARY` vagy `THRESH_BINARY_INV` értékűnek kell lennie,
- `blockSize`: a pixel szomszédságának mérete, amelyet a pixel küszöbértékének kiszámítására használnak,
- `C`: a konstanst az átlagból vagy a súlyozott átlagból kell levonni, normális esetben pozitív, de lehet nulla vagy negatív is.

A maszk színesre konvertálásához a

```
cvtColor(img_edge, img_edge, CV_GRAY2RGB);
```

függvényt használhatjuk, ahol

- `img_edge`: a bemeneti szürkeárnyalatos kép,
- `img_edge`: a kimeneti színes kép,
- `CV_GRAY2RGB`: szürkeárnyalatos kép színesre alakítására használatos paraméter.

Az elmosott képet és a maszkot

```
Mat img_cartoon;
bitwise_and(img_res, img_edge, img_cartoon);
```

programsorokkal egyesíthetjük, ahol

- `img_res`: a bemeneti elmosott kép,
- `img_edge`: a bemeneti élmaszk,
- `img_cartoon`: kimeneti Cartoon-style filterezett kép.

5.3.2. Pencil sketch filter

Első lépésként átkonvertáltam a képet színesről szűrkeárnyalatossá majd egy medián szűrést hajtottam végre. Ezt az előző szűrőnél már bemutattam így itt nem részletezném, annyiban változik a paraméterezése, hogy a kernel mérete itt egy 3×3-as mátrix. A következő lépésben egy Gauss simítást implementáltam.

```
Mat gauss;
Size ksize = Size(21, 21);
double sigmaX = 0;
double sigmaY = 0;
GaussianBlur(median, gauss, ksize, sigmaX, sigmaY);
```

A `GaussianBlur` függvény paraméterezésében

- `median`: a bemeneti medián szűrővel ellátott kép,
- `img_blur`: a kimeneti Gauss szűrés eredménye,
- `ksize`: a kernel mérete, itt ebben az esetben egy 21×21-es mátrix,
- `sigmaX`: a Gauss kernel standard elterese x iranyba,
- `sigmaY`: a Gauss kernel standard elterese y iranyba.

A medián és Gauss szűrő elosztása egymással.

```
Mat img_blend;
double scale_d = 245;
divide(median, gauss, img_blend, scale_d);
```

A `divide` függvényben

- `median`: a bemeneti medián szűrővel ellátott kép,
- `img_blur`: a bemeneti Gauss szűrővel ellátott kép,
- `img_blend`: a kimeneti kép a szűrők elosztásának eredménye,
- `scale_d`: a skalár tényező.

Ezt követően kontraszt széthúzást alkalmaztam az alábbi kóddal.

```
double alpha = 0;
double beta = 255;
normalize(img_blend, img_blend, alpha, beta, CV_MINMAX);
```

A `normalize` függvény paraméterei:

- `img_blend`: a két szűrő elosztásával kapott kép,
- `img_blend`: a kimeneti kontraszt széthúzással kapott kép,
- `alpha`: normál érték a normálértékhez viszonyítva vagy az alsó tartomány határértéke esetén,
- `beta`: első tartományhatár a tartomány normalizálása esetén,
- `CV_MINMAX`: választható maszk művelet.

A szürkeárnyalatos vászon összeszorzása a kontraszt széthúzott képpel való összeszorzása:

```
double scale_m = 1.0 / 256;
multiply(img_blend, canvas, img_blend, scale_m);
```

ahol

- `img_blend`: a bemeneti kontraszt széthúzással ellátott kép,
- `canvas`: a bemenet vászon képe,
- `img_blend`: a kimeneti összeszorzott kép,
- `scale_m`: választható skálafaktor.

5.3.3. Cartoon filter

Első lépésként itt mint az előző filtereknél, átkonvertáltam a képet színesről szürkeárnyalatossá majd egy medián szűrést hajtottam végre, ahol a kernel mérete 7×7-es matrix. Ezt egy Laplace éldetektálás követte.

```
int kernel_size = 5;
Laplacian(median, edges, CV_8U, kernel_size);
```

A `Laplacian` függvény paraméterei:

- `median`: a bemeneti median szűrővel ellátott kép,
- `edges`: az élekdetektálás eredménye,
- `CV_8U`: unsigned 8bit/pixel, azaz egy pixel $[0, 255]$ egész értékű tartományban lehet,
- `kernel_size`: a kernel mérete, itt ebben az esetben egy 5×5-ös mátrix. Azt mutatja meg mekkora legyen a maszkon az él mérete.

Végezetül a küszöbölés következett.

```
double thresh = 90;
double maxval = 255;
threshold(edges, mask, thresh, maxval, THRESH_BINARY_INV);
```

Ebben az

- `edges`: a bemeneti kép az élekdetektálás eredménye,
- `mask`: a küszöbölés eredménye,
- `thresh`: küszöbérték,

- `maxval`: a `THRESH_BINARY` és a `THRESH_BINARY_INV` küszöbérték típusokhoz használható maximális érték,
- `THRESH_BINARY_INV`:

$$mask(x,y) = \begin{cases} 0, & \text{ha } edge(x,y) > thresh, \\ maxval, & \text{különben.} \end{cases}$$

5.3.4. Aquarelle-stlye filter

Ez a filter két egyszerű lépésből áll. Az első az egy átlagoló szűrő.

```
Size ksize = Size(3, 3);
blur(img, avg, ksize);
```

amelyben

- `img`: a bemeneti színes kép,
- `avg`: a kimeneti elmosott kép,
- `ksize`: a kernel mérete, itt ebben az esetben egy 3×3-as mátrix.

Mean shift szegmentáció.

```
Mat img_shifted;
double sp = 15;
double sr = 50;
pyrMeanShiftFiltering(avg, img_shifted, sp, sr);
```

ahol

- `avg`: a bemeneti átlagolással elmosott színes kép,
- `img_shifted`: a kimeneti mean shift szegmentálással előállított kép,
- `sp`: a térbeli ablak sugara,
- `sr`: a színablak sugara.

6. fejezet

Tesztek, eredmények bemutatása

Azokat a szűrőket amelyeket a 4. fejezetben tárgyaltam, nem csak képek átalakítására használtam, hanem videókon is teszteltem. Ebben a fejezetben szeretném leírni ezzel kapcsolatban a tapasztalataimat. Fontosnak tartottam, hogy a vizsgálatok a futási időkre is kitérjenek, ezért a feldolgozási lépések számítási idejét külön-külön lemértem.

A tesztekhez használt számítógép konfigurációja a következő:

- MacBook Pro 2014 mid,
- 2.6 GHz Intel Core i5 processzor,
- 8GB RAM,
- Intel Iris 1536 MB grafikus kártya,
- macOS High Sierra operációs rendszer.

6.1. Szűrők tesztelése képeken

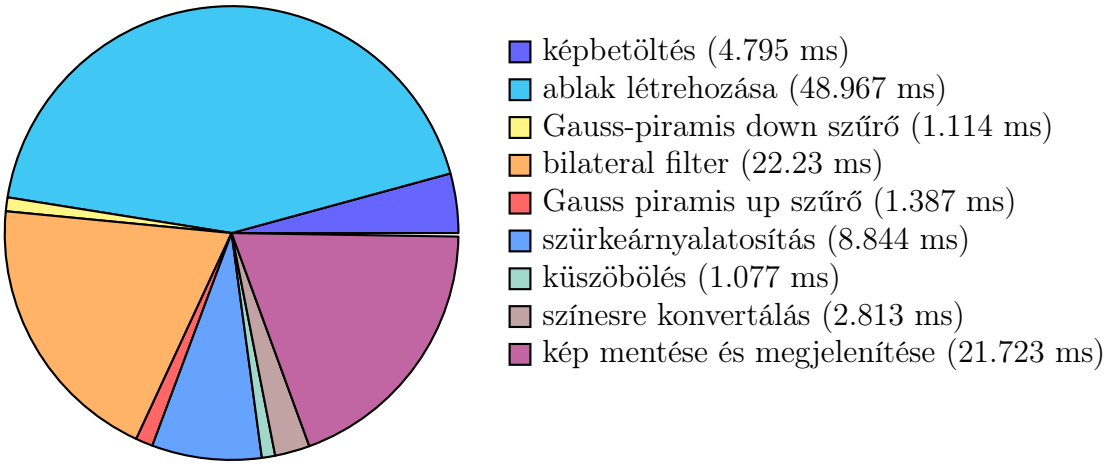
Először képeken teszteltem az elkészített szűrőket. Az első tesztekben azt mutatom meg, hogy mennyi milliszekundum alatt készül el a kép és melyik művelet mennyi ideig tart, szintén milliszekundumban megadva.

A teszteknél a cél az, hogy megmutassa a korábban vizsgált szűrők számítási idejét. Ehhez az eddigi képek kerültek felhasználásra. A bemutatott példák számítási idejeit láthatjuk a következő szakaszokban.

6.1.1. Cartoon-style filter

A szűrő egyes lépéseinek időigényét a 6.1. ábrán láthatjuk. Az aktuális program esetében jól láthatóan időigényes művelet volt a kép betöltése, illetve a megjelenítéshez az ablak létrehozása.

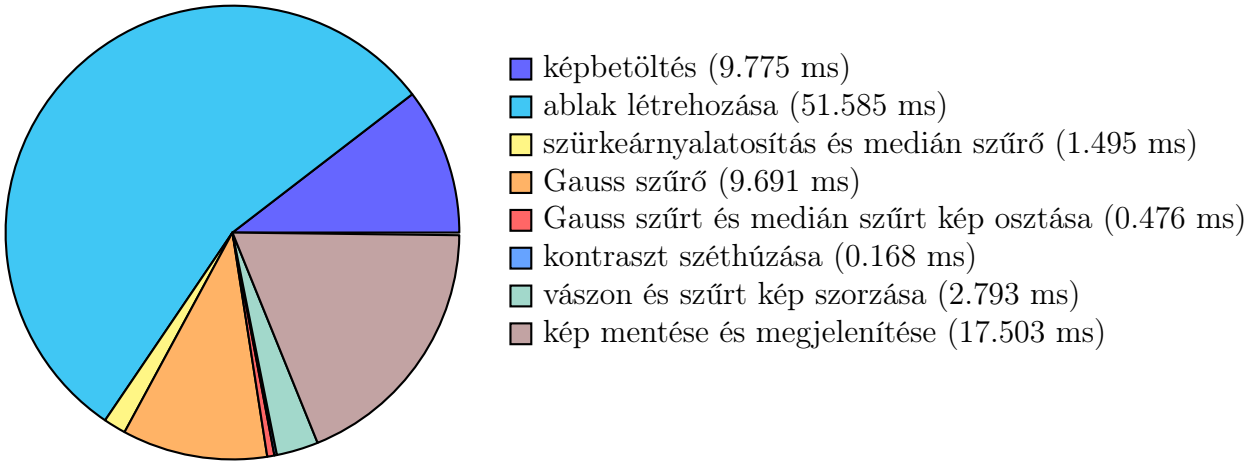
A lényegi lépések közül a bilaterális szűr alkalmazása vette el a legtöbb időt, majd a szürkeárnyalatosra alakítás.



6.1. ábra. Cartoon style filter alkalmazásának időigénye (összesen 113.229 ms)

6.1.2. Pencil sketch filter

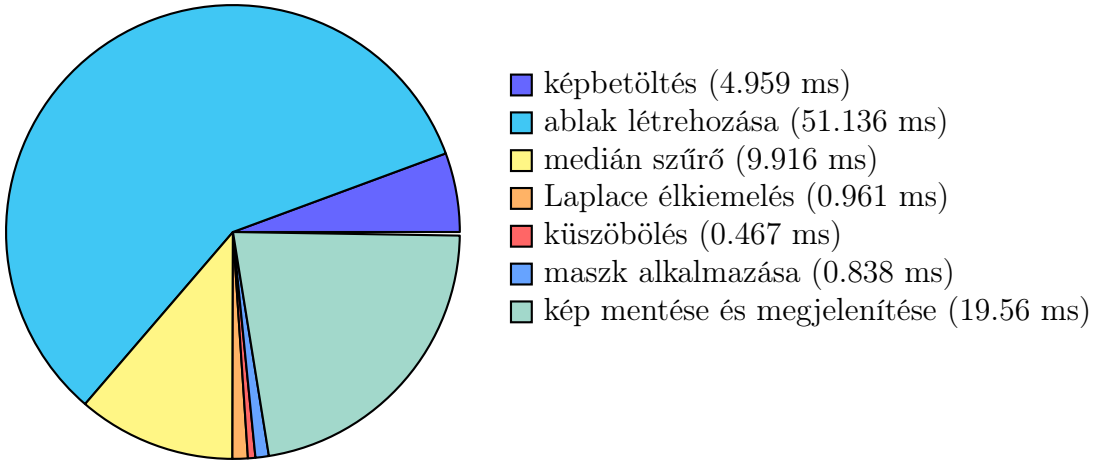
A szűrő szemléltetésére készített alkalmazás futásának jelentős idejét itt is a kép betöltése és az ablak megjelenítése tette ki (6.2. ábra). A következő nagyobb költségű művelet a Gauss szűrő alkalmazása volt.



6.2. ábra. Pencil sketch filter alkalmazásának időigénye (összesen 93.661 ms)

6.1.3. Cartoon filter

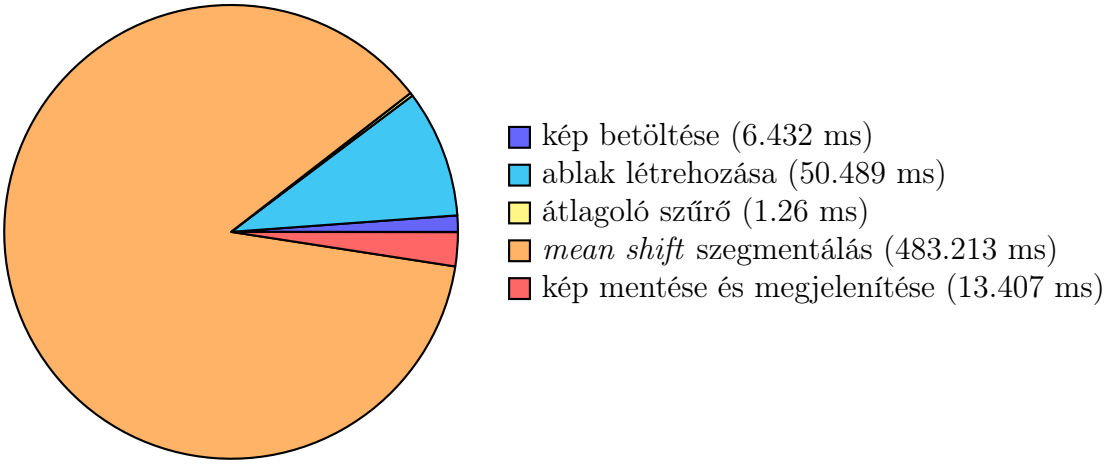
A *Cartoon filter* esetében láthatjuk, hogy a medián szűrés mennyire számításigényes művelet, mivel a kernelen belüli értékek rendezése szükséges hozzá (6.3. ábra). A Laplace élkiemelés, küszöbölés és a maszk alkalmazása is számítási időt tekintve lényegesen kisebb.



6.3. ábra. Cartoon filter alkalmazásának időigénye (összesen 88.059 ms)

6.1.4. Aquarelle-style filter

A vízfestékszerű hatáshoz használt *mean shift* szegmentálás nagyon számításigényes művelet, ahogy az a feldolgozási lépések számítási idejéből látszik (6.4. ábra). Az előzőekhez képest ez a szűrő az, amelyik a legtöbb számítást vette igénybe.



6.4. ábra. Aquarelle-style filter alkalmazásának időigénye (összesen 554.927 ms)

6.2. Szűrők tesztelése videókon és valós időben

Ebben a részben, a videók valamint az élő kép feldolgozási idejét mérem képkocka per másodpercben, azaz fps-ben. Illetve az esetleges képi hibákról és azok kijavításási lehetőségeiről ejtek néhány szót.

6.2.1. Cartoon-style filter

Ha futtatjuk az első filtert, látható, hogy a videó képe vibrál néhány pontban.

Ezeket ki lehet javítani, ha veszünk egy bizonyos kernel méretet és megvizsgáljuk, látjuk hogy a kernelen belül a színek túl nyomó többségben megegyeznek, viszont van néhány pont ami fekete, akkor a fekete pontokat átállítjuk olyan színüre, ami többségben van a kernelen belül. A videóknál, valamint a valós idejű feldolgozásnál mértem képkocka per másodpercet is. Látszik, hogy nem éri el a 24 fps-t sem a videó, sem a kamera képe ennél a szűrőnél. Azért 24 fps mivelt azt már az emberi szem folyamatosnak érzékeli, itt viszont kissé lassabb a kép. A méréseim alapján a videókon átlagosan 15-16 közötti képkocka szám van, a kamera képén 12-15.

6.2.2. Pencil sketch filter

Számításaim alapján ez a szűrő volt az, ami már a videókon majdnem elérte a kellő fps számot, hogy folyamatos legyen. Itt videókon 20-21 között volt a képkocka szám, valós időben pedig 15-16. Itt nem volt annyira sok az egyes műveletek számítási ideje.

6.2.3. Cartoon filter

Erről a szűrőről is elmondható ugyan az mint a Cartoon-style filterről, hogy a kép vibrál. Itt is alkalmazható lehetne az a megoldás, amit ott már leírtam. Itt a képkocka szám másodpercenként a videókban 17-18 között volt, valós időben 13-14 között.

6.2.4. Aquarelle-style filter

Mint látható a képernyőnkön, ha futtatjuk a programot videóval vagy a saját kameránk képével eléggé "szaggat", vagyis a képkocka szám per másodperc, nagyon alacsony. Láthatjuk, hogy az előző képi tesztekben is ennek a szűrőnek volt a legnagobb feldolgozási ideje. A mean shift szegmentáció olyan mértékű számítás igényt jelent a programnak, hogy nem tudja teljesíteni a kívánt fps számot a számítógépem. Itt átlagosan a videón 3,2-3,5 közötti képkocka számot mértem, élő képen 2,5-2,8.

7. fejezet

Összegzés

A dolgozat a művészi szűrők témakörét mutatta be azok matematikai modelljén és néhány saját szűrő megvalósítás segítségével.

A dolgozat megírása előtt még nem foglalkoztam sem képfeldolgozással, sem azokkal az algoritmusokkal amelyeket itt említettem és használtam. Mindig is érdekelt, hogy ezek hogyan működnek. Az első pár hónapban komoly háttérkutatásokat végeztem, mind az algoritmusok és azok matematikai hátterével kapcsolatban.

Mint az előző fejezetekben olvasható, négy saját szűrőt raktam össze amelyek, rajzfilm, ceruzarajz-szerű, továbbá festmény jellegűek. Ezek mindegyikének bemutattam a matematikáját, valamint az implementációját. Néhányat ezek közül, leírások segítségével állítottam össze, de volt olyan amelyek teljesen saját ötlet alapján készült. A filterek algoritmusainak matematikai háttere az előző háttérkutatás után már nem volt ismeretlen, így már csak egy eszköz kellett, amivel meg is lehetett valósítani ezeket.

Korábban nem használtam az OpenCV könyvtárat. A C++ programozási nyelv használata tünt a legcélszerűbbnek. (Eleinte C-ben kezdtem a kódok írását, de több olyan algoritmus implementációja hiányzik az OpenCV könyvtárból ami C++-ban viszont megtalálható. Ezen algoritmusokat viszont fontosak voltak a saját készítésű filterekhez.) Az algoritmusok a könyvtárban könnyedén használhatók, mint ahogy a az 5. fejezetben részletezem. Egyszerűen megadjuk a kívánt paramétereket és már is elértük a kívánt műveletet.

Tesztekkel meg sikerült vizsgálni, hogy a szűrők használata során az egyes lépések számítási ideje milyen. Ezáltal jól láthatóvá váltak a képfeldolgozás szempontjából költségesebb műveletek. A videókon, továbbá a valós időben való képfeldolgozás a saját készítésű szűrők esetén néhány helyen még további kutatást és fejlesztést igényelne. A videók képe vibrál, ahogy a 6. fejezetben is említettem, de akár látható is a dolgozathoz csatolt CD-mellékleten, ha futtatjuk ezen alkalmazásokat. Ezen hibák javítására szerepelnek javaslatok a dolgozatban, viszont javításukhoz egy külön, teljes körű, célzottan erre a problémakörre koncentráló kutatásra lenne szükség.

Summary

This work presents the topic of artistic filters, their mathematical foundations, and some of my own filter implementations.

This is my first time working with image processing algorithms. I was always interested in how they work. In the first couple of months, I did background research about algorithms and their mathematical background. I have shown the results of this research in Chapter 3.

I have designed and implemented four filters for cartoon, pencil and painting-like filtering. These filters take both theoretical and practical aspects into consideration. I have used the available literature, but the filters are based on my original ideas. Background research was conducted, mathematical formulas were given, and appropriate software tools were found for the implementation.

I have never used the OpenCV library before. The usage of the C++ programming language seems to be the proper solution. (Initially, I started to code in C, but some algorithms are implemented in C++, which were necessary for my filters.) The algorithms of the library are easy to use, as we can see in Chapter 5. I have provided the appropriate parameters and reached the desired operation.

I have checked the calculation time of the filtering steps, revealing the time consuming filtering operations. Some aspects of the video and real-time image processing require further research and development. The vibrating noise in the videos (as mentioned in Chapter 6 and can be checked by running the software) should be also filtered. I have proposed some solutions for reducing this type of noise, but their detailed consideration is out of the scope of this work.

Irodalomjegyzék

[1] Papari, Giuseppe, Nicolai Petkov, and Patrizio Campisi. *Artistic edge and corner enhancing smoothing.* IEEE Transactions on Image Processing 16.10 (2007): 2449-2462.

[2] Reese, L. Jack, and William A. Barrett. *Image editing with intelligent paint.* Proceedings of Eurographics. Vol. 21. No. 3. 2002.

[3] Instagram, hivatalos weboldal, `https://www.instagram.com/`, Internet, 2018.

[4] Snapchat, hivatalos weboldal, `https://www.snapchat.com/`, Internet, 2018.

[5] Facebook messenger, hivatalos weboldal, `https://www.messenger.com/features`, Internet, 2018.

[6] Prisma, hivatalos weboldal, `https://prisma-ai.com/`, Internet, 2018.

[7] Pixect, hivatalos weboldal, `https://www.pixect.com/`, Internet, 2018.

[8] Smilebit: *Jet Set Radio*, SEGA, `http://www.sega.com/games/jet-set-radio`, Internet, 2003.

[9] The New York Times, *'A Scanner Darkly': Keanu Reeves, Undercover and Flying High on a Paranoid Head Trip*, 2006. július 7.

[10] Port.hu, *Kamera által homályosan*, `https://port.hu/adatlap/film/tv/kamera-altal-homalyosan-a-scanner-darkly/movie-82437`, Internet, 2006.

[11] Czap László: Képfeldolgozás, egyetemi jegyzet, Miskolci Egyetem, `http://mazsola.iit.uni-miskolc.hu/ czap/HEFOP/Kepfeld1010.pdf`, 2007.

[12] Kató Zoltán: Digitális képfeldolgozás, egyetemi kurzus, Szegedi Tudományegyetem, `http://www.inf.u-szeged.hu/ kato/teaching/DigitalisKepfeldolgozasTG`, 2014.

[13] Paris, Sylvain, et al. *A gentle introduction to bilateral filtering and its applications.* ACM SIGGRAPH 2007 courses. ACM, 2007.

[14] Utkarsh Sinha: *The Sobel and Laplacian Edge Detectors*, http://aishack.in/tutorials/sobel-laplacian-edge-detectors/, 2010.

[15] Michael Beyeler: *How to create a cool cartoon effect with OpenCV and Python*, http://www.askaswiss.com/2016/01/how-to-create-cartoon-effect-opencv-python.html, 2016. január 5.

[16] Michael Beyeler: *How to create a beautiful pencil sketch effect with OpenCV and Python*, http://www.askaswiss.com/2016/01/how-to-create-pencil-sketch-opencv-python.html, 2016. január 13.

[17] Shervin Emami: *Mastering OpenCV with Practical Computer Vision Projects*, Packt Publishing, 2012.

[18] Bradski, Gary, and Adrian Kaehler. Learning OpenCV: *Computer vision with the OpenCV library*. O'Reilly Media, Inc., 2008.

CD-melléklet tartalma

A dolgozat PDF változatát a `dolgozat.pdf` fájlban találjuk.

A dolgozat LaTeX segítségével készült. A forrásfájlok a `dolgozat` jegyzékben találhatók.

A dolgozathoz 4 művészi szűrő került implementálásra. Ezek a `filters` jegyzék alábbi aljegyzékeiben kaptak helyet.

- `cartoon-style`
- `pencil-sketch`
- `cartoon`
- `aquarelle`

A példákban szereplő minták a forrásfájlok mellett találhatók. A program lefordítása után az közvetlenül kipróbálható.